
Running AGP1 iteration 1/10

Author Annotation: Elitism

```
def selection(population, fitnesses):
    """
    Selects the fittest individual from a population.
    Args:
        population: A list of individuals.
        fitnesses: A list of corresponding fitness values.
    Returns:
        The fittest individual in the population.
    """

    # Combine each individual with its fitness value into a list of tuples
    population_with_fitness = list(zip(population, fitnesses))

    # Sort this combined list by the fitness value (i.e., minimize errors)
    sorted_population = sorted(population_with_fitness, key=lambda x: sum(x[1]))

    # The fittest individual is now at the beginning of the sorted list
    return sorted_population[0][0]
```

Running AGP1 iteration 2/10

Author Annotation: Elitism

```
def selection(population, fitnesses):
    """Selection function for Genetic Programming."""

    # Combine population and fitnesses into a list of tuples
    pop_fitness = list(zip(population, fitnesses))

    # Sort the list in ascending order based on fitness (fewer errors is better)
    sorted_pop = sorted(pop_fitness, key=lambda x: np.sum(x[1]))

    # Select the best individual with the lowest error sum as the parent
    parent = sorted_pop[0][0]

    return parent
```

Running AGP1 iteration 3/10

Author Annotation: Elitism

```
def selection(population, fitnesses):
    # Calculate the total errors for each individual in the population
    total_errors = [np.sum(fitness) for fitness in fitnesses]

    # Select the index of the individual with the lowest total errors (i.e., highest fitness)
```

```
    idx_best_individual = np.argmin(total_errors)

    return population[idx_best_individual]
```

Running AGP1 iteration 4/10

Author Annotation: Elitism

```
def selection(population, fitnesses):
    """
    Parent Selection Function for Genetic Programming.

    This function takes a population of individuals and their corresponding fitness values,
    both given as lists. It returns one individual from the population.

    The GP system used here has tree-based representation with 'add', 'sub', and 'mul' as non-terminals
    and 'x', -2, -1, 0, 1, and 2 as terminals.

    Parameters:
    population (list): A list of individuals.
    fitnesses (list): A list of fitness values corresponding to each individual in the population.

    Returns:
    An individual from the population.
```

```
    """

    # Combine population and fitness lists into a list of tuples for easier manipulation
    population_fitness = list(zip(population, fitnesses))

    # Sort the population based on their fitness (lower is better)
    sorted_population = sorted(population_fitness, key=lambda x: sum(x[1]))

    # Select the individual with the best fitness as the parent
    parent = sorted_population[0][0]

    return parent
```

Running AGP1 iteration 5/10

Author Annotation: Elitism

```
import numpy as np
def selection(population, fitnesses):
    # Combine population and fitnesses into a list of tuples for easier indexing
    combined = list(zip(population, fitnesses))

    # Sort the combined list based on fitness (i.e., errors)
    sorted_combined = sorted(combined, key=lambda x: np.mean(x[1]))
```

```
    # Select the fittest individual(s) with the lowest mean error
    fittest_individuals = [x[0] for x in sorted_combined[:2]]

    # If there's only one fittest individual (i.e., multiple individuals have the same minimum fitness),
    # return this individual; otherwise, choose one of them at random.
    if len(fittest_individuals) == 1:
        selected = fittest_individuals[0]
    else:
        selected = np.random.choice(fittest_individuals)

    return selected
```

Running AGP1 iteration 6/10

Author Annotation: Elitism

```
def selection(population, fitnesses):
    # Calculate average fitness for each individual to select top-performing individuals
    avg_fitness = [sum(f) / len(f) if f else float('inf') for f in fitnesses]

    # Select the individual with the minimum average error (i.e., maximum fitness)
    selected_idx = np.argmin(avg_fitness)

    return population[selected_idx]
```

Running AGP1 iteration 7/10

Author Annotation: Elitism

```
import numpy as np
def selection(population, fitnesses):
    """
    Parent Selection Function for Genetic Programming.
    This function takes a population of individuals and their corresponding
    fitness values as input, and returns one individual from the population.
    The returned individual is chosen based on its fitness value, with the
    best-fit individual being selected.
    Parameters:
    - population (list): A list of individuals in the current population.
    - fitnesses (list): A list of fitness values corresponding to each individual
       in the population.
    Returns:
    - individual: An individual from the population, chosen based on its fitness.
    """

    # Combine the population and fitness lists into a single list of tuples,
    # where each tuple contains an individual and its corresponding fitness value.
    combined = [(individual, fitness) for individual, fitness in zip(population, fitnesses)]

    # Sort the combined list in ascending order based on the fitness values.
```

```
    # The sorted function in Python uses Timsort, which has a worst-case time
    # complexity of O(n log n).
    combined.sort(key=lambda x: np.mean(x[1]))  # np.mean used to handle lists of errors

    # Select the first individual from the sorted list as the parent for crossover.
    # This is because in this implementation, we are trying to minimize the fitness
    # (i.e., maximize the accuracy) and the best-fit individuals have lower fitness.
    parent = combined[0][0]

    return parent
```

Running AGP1 iteration 8/10

Author Annotation: Elitism

```
def selection(population, fitnesses):
    """
    Parent selection for Genetic Programming.
   Parameters:
    population (list): list of individuals
    fitnesses (list): corresponding list of fitness values
    Returns:
    individual: selected parent from the population
    """
```

```
    # Calculate total errors for each individual in the population
    total_errors = [np.sum(fitness) for fitness in fitnesses]

    # Select the index of the individual with the lowest total error
    min_error_idx = np.argmin(total_errors)

    return population[min_error_idx]
```

Running AGP1 iteration 9/10

Author Annotation: Elitism

```
def selection(population, fitnesses):
    """
    Selects a parent from the population based on their fitness.
    Parameters:
    population (list): A list of individuals in the population.
    fitnesses (list): A list of fitness values corresponding to each individual in the population.
    Returns:
    An individual selected as the parent for reproduction.
    """

    # Combine the population and fitnesses into a single data structure
    combined = list(zip(population, fitnesses))

    # Sort the individuals based on their fitness (lower is better)
```

```
    combined.sort(key=lambda x: np.sum(x[1]))

    # Select the best individual as the parent
    # The best individual has the minimum total error
    selected_parent = [x for x, y in combined if np.sum(y) == min([np.sum(z) for _, z in combined])][0]

    return selected_parent
```

Running AGP1 iteration 10/10

Author Annotation: Fitness-proportionate-esque

```
def selection(population, fitnesses):
    """
    Select a parent from the population based on their fitness.
    Parameters:
    population (list): A list of individuals.
    fitnesses (list): A list of fitness values corresponding to each individual in the population.
    Returns:
    individual: The selected individual with the best fitness.
    """

    # Calculate cumulative probabilities for selection by rank
    cum_prob = [0] * len(population)
    total_fitness = 0
```

```
    for i in range(len(population)):
        # Use negative of error as fitness value because we want to minimize errors
        fitness = -fitnesses[i][0]
        total_fitness += fitness
        if i == 0:
            cum_prob[i] = fitness
        else:
            cum_prob[i] = cum_prob[i-1] + fitness

    # Normalize cumulative probabilities
    for i in range(len(population)):
        cum_prob[i] /= total_fitness

    # Generate a random number between 0 and 1 to select the parent based on cumulative probability
    r = np.random.rand()

    # Find the index of the selected individual with the best fitness
    idx = -1
    for i in range(len(population)):
        if cum_prob[i] >= r:
            idx = i
            break

    return population[idx]
```