```python
import numpy as np
import random

# Define the Rastrigin function (a well-known benchmark for optimization)
def rastrigin(x):
    A = 10
    return A * len(x) + sum([(xi**2 - A * np.cos(2 * np.pi * xi)) for xi in x])

# Initialize population
def initialize_population(pop_size, num_genes, lower_bound, upper_bound):
    population = np.random.uniform(lower_bound, upper_bound, (pop_size, num_genes))
    return population

# Evaluate fitness of the population
def evaluate_fitness(population):
    fitness = np.array([rastrigin(individual) for individual in population])
    return fitness

# Selection: Tournament selection
def tournament_selection(population, fitness, tournament_size=3):
    selected = []
    for _ in range(len(population)):
        tournament_indices = np.random.choice(len(population), tournament_size, replace=False)
        tournament_fitness = fitness[tournament_indices]
        winner_idx = tournament_indices[np.argmin(tournament_fitness)] # Minimize the Rastrigin function
        selected.append(population[winner_idx])
    return np.array(selected)

# Crossover: One-point crossover
def crossover(parent1, parent2):
    crossover_point = np.random.randint(1, len(parent1) - 1)
    child1 = np.concatenate((parent1[:crossover_point], parent2[crossover_point:]))
    child2 = np.concatenate((parent2[:crossover_point], parent1[crossover_point:]))
    return child1, child2

# Mutation: Random mutation
def mutate(child, mutation_rate, lower_bound, upper_bound):
    for i in range(len(child)):
        if np.random.rand() < mutation_rate:
            child[i] = np.random.uniform(lower_bound, upper_bound)
    return child

# Gene expression (mapping genes to real values, already done by direct mapping in this case)
# You can modify this step based on the problem's domain (i.e., gene representation and translation)
# Main GEA function
def gene_expression_algorithm(pop_size, num_genes, lower_bound, upper_bound, mutation_rate, crossover_rate, num_generations):
    # Step 1: Initialize Population
    population = initialize_population(pop_size, num_genes, lower_bound, upper_bound)

    # Step 2: Iterate for a fixed number of generations
    best_solution = None
    best_fitness = float('inf')

    for generation in range(num_generations):
        # Step 3: Evaluate fitness
        fitness = evaluate_fitness(population)

        # Step 4: Track the best solution
        min_fitness_idx = np.argmin(fitness)
        if fitness[min_fitness_idx] < best_fitness:
            best_fitness = fitness[min_fitness_idx]
            best_solution = population[min_fitness_idx]

        # Step 5: Selection
        selected_population = tournament_selection(population, fitness)
```

```python
        # Step 6: Crossover and Mutation
        new_population = []
        for i in range(0, pop_size, 2):
            parent1 = selected_population[i]
            parent2 = selected_population[i+1] if i+1 < pop_size else selected_population[0] # Ensuring even number of parents

            # Perform crossover
            if np.random.rand() < crossover_rate:
                child1, child2 = crossover(parent1, parent2)
            else:
                child1, child2 = parent1, parent2 # No crossover, just pass parents

            # Apply mutation
            child1 = mutate(child1, mutation_rate, lower_bound, upper_bound)
            child2 = mutate(child2, mutation_rate, lower_bound, upper_bound)

            # Add the children to the new population
            new_population.extend([child1, child2])

        # Update population with new generation
        population = np.array(new_population[:pop_size]) # Ensure population size remains constant

    return best_solution, best_fitness

# Set parameters
pop_size = 100                  # Population size
num_genes = 10                  # Number of genes (dimensions of the problem)
lower_bound = -5.12             # Lower bound of the search space
upper_bound = 5.12              # Upper bound of the search space
mutation_rate = 0.1             # Mutation rate
crossover_rate = 0.8            # Crossover rate
num_generations = 500           # Number of generations

# Run the Gene Expression Algorithm
best_solution, best_fitness = gene_expression_algorithm(pop_size, num_genes, lower_bound, upper_bound, mutation_rate, crossover_rate, num_generations)

# Output the results
print("Best Solution:", best_solution)
print("Best Fitness:", best_fitness)
```

```
Best Solution: [ 0.01956405  0.00271381 -0.00243719  0.00141388 -0.02586832  0.00105932
  0.01769152 -1.03340239 -0.02943199 -0.04696745]
Best Fitness: 2.166804134722355
```