```
import numpy as np

# Objective Function (to be optimized)
def objective_function(x):
    # Example: simple sum of squares (minimization problem)
    return np.sum(x**2)

# Generate a random Lévy flight step
def levy_flight(D, alpha=1.0):
    # Generate random step based on Lévy flight distribution (simplified)
    u = np.random.normal(0, 1, D)  # Normal distributed random variable u
    v = np.random.normal(0, 1, D)  # Normal distributed random variable v
    step = u / (np.abs(v) ** (1 / alpha))  # Lévy flight step
    return step

# Random initialization of nests in a D-dimensional space
def randomly_initialize(N, D, lower_bound, upper_bound):
    return np.random.uniform(lower_bound, upper_bound, (N, D))

# Evaluate the fitness of each nest
def evaluate_fitness(nests, func):
    return np.apply_along_axis(func, 1, nests)

# Find the index of the nest with the minimum fitness
def min_fitness_index(fitness):
    return np.argmin(fitness)

# Get the worst nests based on fitness
def get_worst_nests(fitness, pa, N):
    sorted_indices = np.argsort(fitness)
    worst_indices = sorted_indices[-int(pa * N):]
    return worst_indices

# Replace the worst nests with random new nests
def replace_worst_nests(nests, worst_nests, N, D, lower_bound, upper_bound):
    for i in worst_nests:
        nests[i] = np.random.uniform(lower_bound, upper_bound, D)
    return nests

# Cuckoo Search Algorithm
def cuckoo_search(Func, D, N, MaxIter, pa, alpha, lower_bound, upper_bound):
    # Initialize nests randomly in D-dimensional space
    nests = randomly_initialize(N, D, lower_bound, upper_bound)

    # Evaluate fitness of each nest
    fitness = evaluate_fitness(nests, Func)

    # Find the best nest so far
    best_nest = nests[min_fitness_index(fitness)]
    best_fitness = np.min(fitness)

    # Main optimization loop
    for iteration in range(MaxIter):
        # Generate new solutions using Lévy flights
        for i in range(N):
            step_size = alpha * levy_flight(D)
            new_nest = nests[i] + step_size
            # Ensure the new nest is within bounds
            new_nest = np.clip(new_nest, lower_bound, upper_bound)
            new_fitness = Func(new_nest)

            # If new nest is better, replace it
            if new_fitness < fitness[i]:
                nests[i] = new_nest
                fitness[i] = new_fitness

        # Abandon worst nests (replace with random new nests)
        worst_nests = get_worst_nests(fitness, pa, N)
        nests = replace_worst_nests(nests, worst_nests, N, D, lower_bound, upper_bound)

        # Update the best nest if needed
        current_best_index = min_fitness_index(fitness)
        if fitness[current_best_index] < best_fitness:
            best_nest = nests[current_best_index]
            best_fitness = fitness[current_best_index]
```

```python
    return best_nest, best_fitness


# Example usage
if __name__ == "__main__":
    # Problem Parameters
    D = 5  # Dimensionality of the problem
    N = 20  # Number of nests
    MaxIter = 100  # Maximum number of iterations
    pa = 0.25  # Probability of nest abandonment
    alpha = 1.0  # Scaling factor for Lévy flight
    lower_bound = -10  # Lower bound of the search space
    upper_bound = 10  # Upper bound of the search space

    # Running Cuckoo Search
    best_nest, best_fitness = cuckoo_search(objective_function, D, N, MaxIter, pa, alpha, lower_bound, upper_bound)

    print("Best solution:", best_nest)
    print("Best fitness:", best_fitness)
```

```
Best solution: [-0.1894033  -0.01304134  0.65105524  0.0616087   0.32599543]
Best fitness: 0.5699852621691621
```

```python
import numpy as np
import random
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris

# 1. Cuckoo Search Algorithm

# Initialize parameters
class CuckooSearch:
    def __init__(self, num_nests, max_iter, pa, alpha=0.01):
        self.num_nests = num_nests
        self.max_iter = max_iter
        self.pa = pa  # Probability of discovery
        self.alpha = alpha  # Step size
        self.nests = None

    def initialize_nests(self, n_features):
        # Initialize nests with random feature subsets
        self.nests = np.random.randint(0, 2, (self.num_nests, n_features))

    def fitness(self, subset, X_train, y_train):
        # Create a subset of the dataset based on feature selection
        selected_features = np.where(subset == 1)[0]

        if len(selected_features) == 0:
            return -1  # Penalize if no feature is selected

        X_subset = X_train[:, selected_features]

        # Train a model (Random Forest in this case)
        clf = RandomForestClassifier()
        clf.fit(X_subset, y_train)
        predictions = clf.predict(X_subset)

        # Return the accuracy as the fitness
        return accuracy_score(y_train, predictions)

    def levy_flight(self, current_nest, best_nest):
        # Generate new solution using Levy flight
        s = np.random.normal(0, 1, current_nest.shape)
        return np.clip(current_nest + self.alpha * s, 0, 1)

    def cuckoo_search(self, X_train, y_train):
        n_features = X_train.shape[1]
        self.initialize_nests(n_features)

        # Evaluate initial nests
        fitness_values = np.array([self.fitness(nest, X_train, y_train) for nest in self.nests])
        best_nest = self.nests[np.argmax(fitness_values)]
        best_fitness = np.max(fitness_values)
```

```
        for iteration in range(self.max_iter):
            # Generate new nests and evaluate them
            for i in range(self.num_nests):
                # Generate new nest based on Levy flight
                new_nest = self.levy_flight(self.nests[i], best_nest)
                new_fitness = self.fitness(new_nest, X_train, y_train)

                # Replace nest if new solution is better
                if new_fitness > fitness_values[i]:
                    self.nests[i] = new_nest
                    fitness_values[i] = new_fitness

                    # Update the best nest found
                    if new_fitness > best_fitness:
                        best_nest = new_nest
                        best_fitness = new_fitness

            # Discovery probability: Replace worst nests with new ones
            for i in range(self.num_nests):
                if random.random() < self.pa:
                    self.nests[i] = np.random.randint(0, 2, n_features)

        return best_nest, best_fitness

# 2. Main Feature Selection with Cuckoo Search

# Load the dataset (using the iris dataset as an example)
data = load_iris()
X = data.data
y = data.target

# Normalize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# Set up the Cuckoo Search Algorithm
cuckoo = CuckooSearch(num_nests=20, max_iter=50, pa=0.25)

# Perform feature selection
best_nest, best_fitness = cuckoo.cuckoo_search(X_train, y_train)

# Print the best feature subset and the fitness
selected_features = np.where(best_nest == 1)[0]
print("Selected features: ", selected_features)
print("Best fitness (accuracy): ", best_fitness)

# Evaluate the model using the selected features
X_train_selected = X_train[:, selected_features]
X_test_selected = X_test[:, selected_features]

clf = RandomForestClassifier()
clf.fit(X_train_selected, y_train)
y_pred = clf.predict(X_test_selected)

# Final model performance
accuracy = accuracy_score(y_test, y_pred)
print("Test accuracy with selected features: ", accuracy)
```

```
Selected features:  [1 2]
Best fitness (accuracy):  1.0
Test accuracy with selected features:  0.9333333333333333
```

Start coding or generate with AI.