In [24]:

```
import matplotlib.pyplot as plt
import mlrose_hiive
import numpy as np
import pandas as pd
import time
import warnings

from sklearn.metrics import accuracy_score
from sklearn.model_selection import (GridSearchCV, train_test_split, valic
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn import preprocessing
np.random.seed(42)
```

Four Peaks

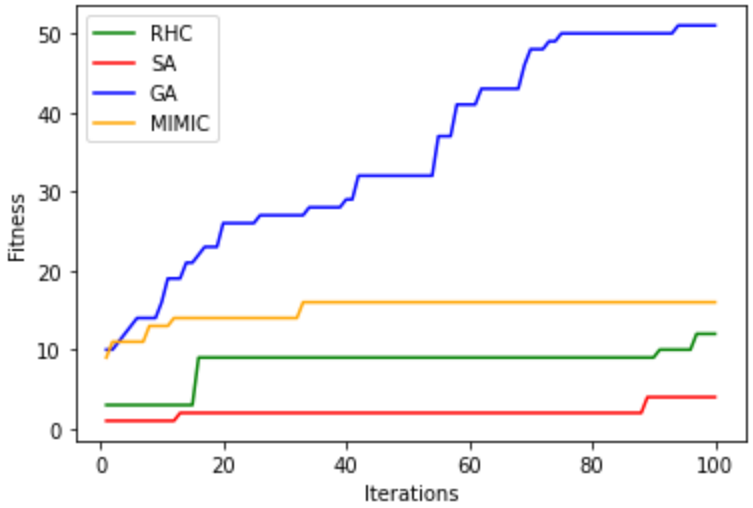
In [35]:

```
print("Running Experiments for Four Peaks Problem")
print()
np.random.seed(0)
# Define Fitness function and discrete problem object
fitness = mlrose_hiive.FourPeaks()
problem = mlrose_hiive.DiscreteOpt(length=100, fitness_fn=fitness, maximiz

max_attempts = 100
max_iters = 100

# RHC
print("Running Random Hill Climb Experiment")
start_time = time.time()
rhc_best_state, rhc_best_fitness, rhc_fitness_curve = mlrose_hiive.random_

end_time = time.time()
rhc_time = end_time - start_time
print("Time (s): {}".format(rhc_time))
print()

# SA
print("Running Simulated Annealing Experiment")
start_time = time.time()
sa_best_state, sa_best_fitness, sa_fitness_curve = mlrose_hiive.simulated_
                                                    problem,
                                                    max_attempts=max_atter
                                                    max_iters=max_iters,
                                                    curve=True,
                                                    random_state=42,
                                                    schedule=mlrose_hiive.
end_time = time.time()
sa_time = end_time - start_time
print("Time (s): {}".format(sa_time))
print()

# GA
print("Running Genetic Algorithm Experiment")
start_time = time.time()
ga_best_state, ga_best_fitness, ga_fitness_curve = mlrose_hiive.genetic_al
                                                    problem,
                                                    max_attempts=max_atter
                                                    max_iters=max_iters,
                                                    curve=True,
                                                    random_state=42,
                                                    pop_size=100,
                                                    mutation_prob=0.4)
end_time = time.time()
ga_time = end_time - start_time
print("Time (s): {}".format(ga_time))
print()

# MIMIC
print("Running MIMIC Algorithm Experiment")
```

```
start_time = time.time()
mimic_best_state, mimic_best_fitness, mimic_fitness_curve = mlrose_hiive.r
                                                            problem,
                                                            max_attempts =
                                                            max_iters = 10
                                                            curve = True,
                                                            random_state =
                                                            keep_pct=0.5)
end_time = time.time()
mimic_time = end_time - start_time
print("Time (s): {}".format(mimic_time))
print()
```

```
Running Experiments for Four Peaks Problem

Running Random Hill Climb Experiment
Time (s): 0.12115812301635742

Running Simulated Annealing Experiment
Time (s): 0.0

Running Genetic Algorithm Experiment
Time (s): 1.0939412117004395

Running MIMIC Algorithm Experiment
Time (s): 690.7917082309723
```

In [36]:
```
# Plot Iterations vs Fitness
iterations = range(1, max_iters + 1)
plt.plot(iterations, rhc_fitness_curve[:, 0], label='RHC', color='green')
plt.plot(iterations, sa_fitness_curve[:, 0], label='SA', color='red')
plt.plot(iterations, ga_fitness_curve[:, 0], label='GA', color='blue')
plt.plot(iterations, mimic_fitness_curve[:, 0], label='MIMIC', color='oran
plt.legend(loc="best")
plt.xlabel("Iterations")
plt.ylabel("Fitness")

# Plot Time Table
data = [('RHC', round(rhc_time, 5)),
        ('SA', round(sa_time, 5)),
        ('GA', round(ga_time, 5)),
        ('MIMIC', round(mimic_time, 5))]
```



In [37]:
```
# Plot Time Table
data = [('RHC', round(rhc_time, 5)),
        ('SA', round(sa_time, 5)),
        ('GA', round(ga_time, 5)),
        ('MIMIC', round(mimic_time, 5))]

time_FourPeaks = pd.DataFrame(data, columns =['Algorithm', 'Time (s)'])
time_FourPeaks
```

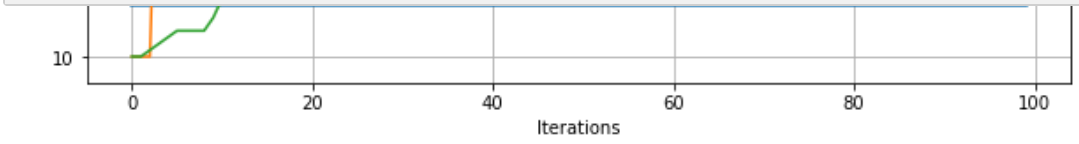
Out[37]:

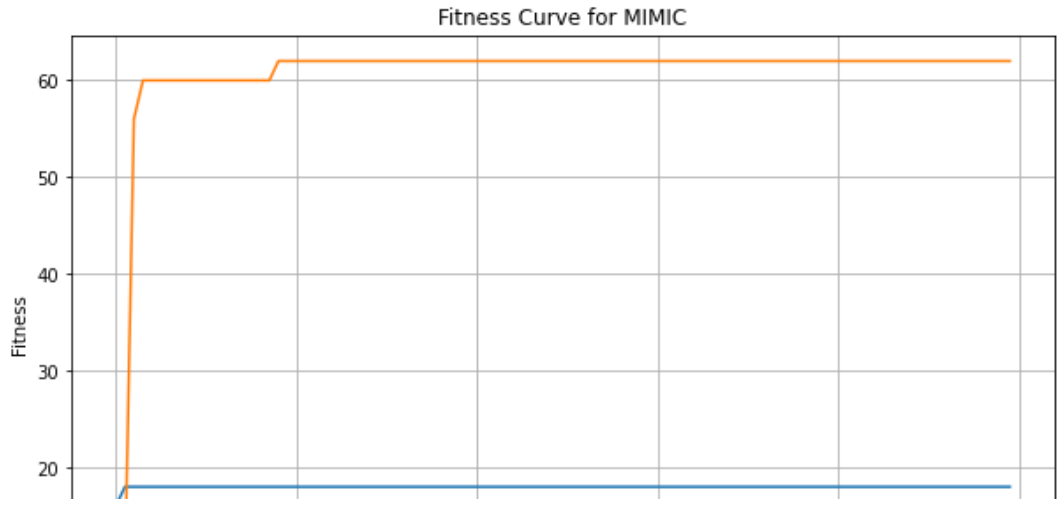
	Algorithm	Time (s)
0	RHC	0.12116
1	SA	0.00000
2	GA	1.09394
3	MIMIC	690.79171

In [38]:

```
np.random.seed(0)

# Define a range of problem sizes (lengths)
problem_sizes = [10, 50, 100]

# Lists to store fitness curves for each algorithm and problem size
fitness_curves = {
    'rhc': [],
    'sa': [],
    'ga': [],
    'mimic': []
}

function_calls = {
    'rhc': [],
    'sa': [],
    'ga': [],
    'mimic': []
}

# Run experiments for each problem size
for length in problem_sizes:
    print("Running experiments for problem size: {}".format(length))

    # Define Fitness function and discrete problem object for the current
    fitness = mlrose_hiive.FourPeaks()
    problem = mlrose_hiive.DiscreteOpt(length=length, fitness_fn=fitness,

    # RHC
    rhc_fitness_curve = mlrose_hiive.random_hill_climb(problem,
                                                       max_attempts=max_at
                                                       max_iters=max_iters
                                                       curve=True,
                                                       random_state=42,
                                                       restarts=75)[2]
    fitness_curves['rhc'].append(rhc_fitness_curve)
    function_calls['rhc'].append(len(rhc_fitness_curve))


    # SA
    sa_fitness_curve = mlrose_hiive.simulated_annealing(problem,
                                                        max_attempts=max_at
                                                        max_iters=max_iters
                                                        curve=True,
                                                        random_state=42,
                                                        schedule=mlrose_hi
    fitness_curves['sa'].append(sa_fitness_curve)
    function_calls['sa'].append(len(sa_fitness_curve))

    # GA
    ga_fitness_curve = mlrose_hiive.genetic_alg(problem,
                                                max_attempts=max_attempts,
                                                max_iters=max_iters,
                                                curve=True,
                                                random_state=42,
                                                pop_size=100,
```

```
                                                    mutation_prob=0.4)[2]
    fitness_curves['ga'].append(ga_fitness_curve)
    function_calls['ga'].append(len(ga_fitness_curve))

    # MIMIC
    mimic_fitness_curve = mlrose_hiive.mimic(problem,
                                             max_attempts=100,
                                             max_iters=100,
                                             curve=True,
                                             random_state=42,
                                             keep_pct=0.5)[2]
    fitness_curves['mimic'].append(mimic_fitness_curve)
    function_calls['mimic'].append(len(mimic_fitness_curve))

# Plot fitness curves for each algorithm and problem size
for algo, curves in fitness_curves.items():
    plt.figure(figsize=(10, 6))
    for i, curve in enumerate(curves):
        plt.plot(curve[:,0], label=f'Problem size: {problem_sizes[i]}')
    plt.xlabel('Iterations')
    plt.ylabel('Fitness')
    plt.title(f'Fitness Curve for {algo.upper()}')
    plt.legend()
    plt.grid(True)
    plt.show()

for algo, calls in function_calls.items():
    plt.figure(figsize=(10, 6))
    plt.plot(calls, marker='o')
    plt.xlabel('Iterations')
    plt.ylabel('Function Calls')
    plt.title(f'Function Calls vs Problem Size for {algo.upper()}')
    plt.grid(True)
    plt.show()
```





Continuous Peaks

In [26]:

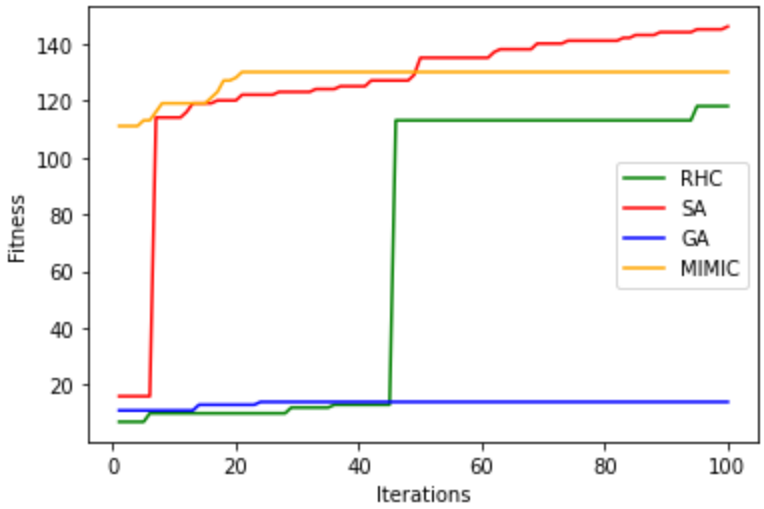
```
print("Running Experiments for Continuous Peaks Problem")
print()
np.random.seed(0)

# Define Fitness function and discrete problem object
fitness = mlrose_hiive.ContinuousPeaks()
problem = mlrose_hiive.DiscreteOpt(length=100, fitness_fn=fitness, maximiz

max_attempts = 100
max_iters = 100

# RHC
print("Running Random Hill Climb Experiment")
start_time = time.time()
rhc_best_state, rhc_best_fitness, rhc_fitness_curve = mlrose_hiive.random_

end_time = time.time()
rhc_time = end_time - start_time
print("Time (s): {}".format(rhc_time))
print()

# SA
print("Running Simulated Annealing Experiment")
start_time = time.time()
sa_best_state, sa_best_fitness, sa_fitness_curve = mlrose_hiive.simulated_
                                                    problem,
                                                    max_attempts=max_atter
                                                    max_iters=max_iters,
                                                    curve=True,
                                                    random_state=42,
                                                    schedule=mlrose_hiive.
end_time = time.time()
sa_time = end_time - start_time
print("Time (s): {}".format(sa_time))
print()

# GA
print("Running Genetic Algorithm Experiment")
start_time = time.time()
ga_best_state, ga_best_fitness, ga_fitness_curve = mlrose_hiive.genetic_al
                                                    problem,
                                                    max_attempts=max_atter
                                                    max_iters=max_iters,
                                                    curve=True,
                                                    random_state=42,
                                                    pop_size=200,
                                                    mutation_prob=0.4)
end_time = time.time()
ga_time = end_time - start_time
print("Time (s): {}".format(ga_time))
print()

# MIMIC
```

```
print("Running MIMIC Algorithm Experiment")
start_time = time.time()
mimic_best_state, mimic_best_fitness, mimic_fitness_curve = mlrose_hiive.r
                                                            problem,
                                                            max_attempts =
                                                            max_iters = 10
                                                            curve = True,
                                                            random_state =
                                                            keep_pct=0.5)
end_time = time.time()
mimic_time = end_time - start_time
print("Time (s): {}".format(mimic_time))
print()
```

```
Running Experiments for Continuous Peaks Problem

Running Random Hill Climb Experiment
Time (s): 1.3980448246002197

Running Simulated Annealing Experiment
Time (s): 0.04026985168457031

Running Genetic Algorithm Experiment
Time (s): 4.612846374511719

Running MIMIC Algorithm Experiment
Time (s): 1182.0504133701324
```

In [28]:
```
# Plot Iterations vs Fitness
iterations = range(1, max_iters + 1)
plt.plot(iterations, rhc_fitness_curve[:, 0], label='RHC', color='green')
plt.plot(iterations, ga_fitness_curve[:, 0], label='SA', color='red')
plt.plot(iterations, sa_fitness_curve[:, 0], label='GA', color='blue')
plt.plot(iterations, mimic_fitness_curve[:, 0], label='MIMIC', color='orar
plt.legend(loc="best")
plt.xlabel("Iterations")
plt.ylabel("Fitness")
```

Out[28]:
```
Text(0, 0.5, 'Fitness')
```



In [29]:
```
# Plot Time Table
data = [('RHC', round(rhc_time, 5)),
        ('SA', round(sa_time, 5)),
        ('GA', round(ga_time, 5)),
        ('MIMIC', round(mimic_time, 5))]

time_ContPeaks = pd.DataFrame(data, columns =['Algorithm', 'Time (s)'])
time_ContPeaks
```

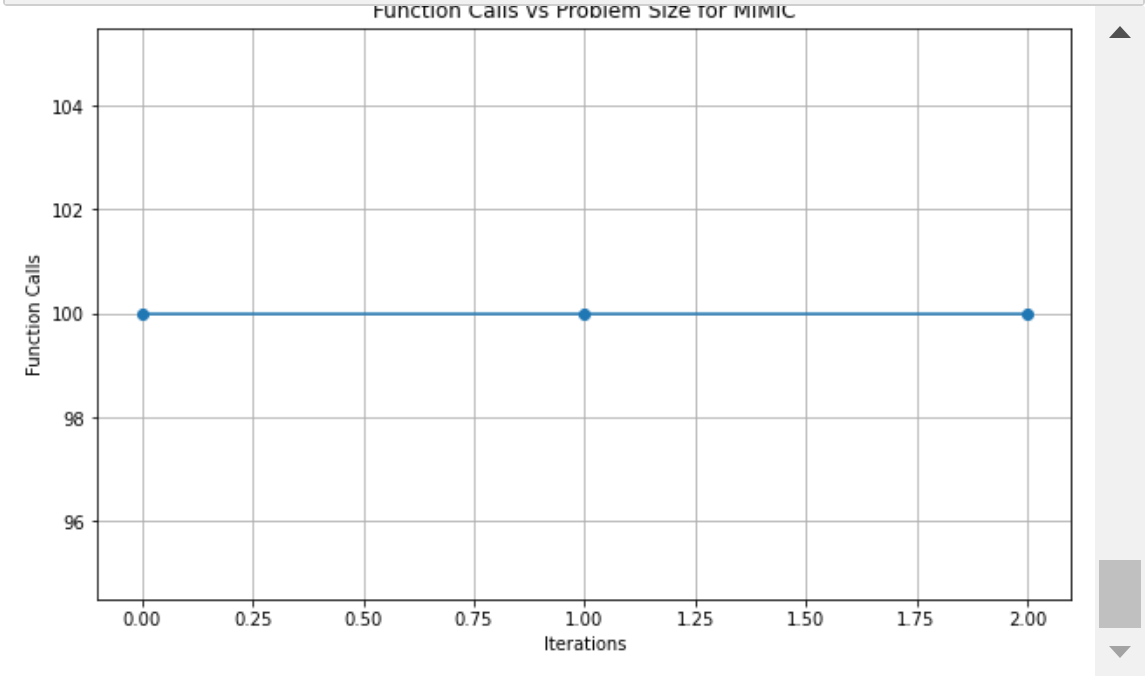
Out[29]:

	Algorithm	Time (s)
0	RHC	1.39804
1	SA	0.04027
2	GA	4.61285
3	MIMIC	1182.05041

In [30]:

```
np.random.seed(0)

# Define a range of problem sizes (lengths)
problem_sizes = [10, 50, 100]

# Lists to store fitness curves for each algorithm and problem size
fitness_curves = {
    'rhc': [],
    'sa': [],
    'ga': [],
    'mimic': []
}

function_calls = {
    'rhc': [],
    'sa': [],
    'ga': [],
    'mimic': []
}

# Run experiments for each problem size
for length in problem_sizes:
    print("Running experiments for problem size: {}".format(length))

    # Define Fitness function and discrete problem object for the current
    fitness = mlrose_hiive.ContinuousPeaks()
    problem = mlrose_hiive.DiscreteOpt(length=length, fitness_fn=fitness,

    # RHC
    rhc_fitness_curve = mlrose_hiive.random_hill_climb(problem,
                                                       max_attempts=max_at
                                                       max_iters=max_iters
                                                       curve=True,
                                                       random_state=42,
                                                       restarts=100)[2]
    fitness_curves['rhc'].append(rhc_fitness_curve)
    function_calls['rhc'].append(len(rhc_fitness_curve))


    # SA
    sa_fitness_curve = mlrose_hiive.simulated_annealing(problem,
                                                        max_attempts=max_at
                                                        max_iters=max_iters
                                                        curve=True,
                                                        random_state=42,
                                                        schedule=mlrose_hi
    fitness_curves['sa'].append(sa_fitness_curve)
    function_calls['sa'].append(len(sa_fitness_curve))

    # GA
    ga_fitness_curve = mlrose_hiive.genetic_alg(problem,
                                                max_attempts=max_attempts,
                                                max_iters=max_iters,
                                                curve=True,
                                                random_state=42,
                                                pop_size=200,
```

```
                                                    mutation_prob=0.4)[2]
    fitness_curves['ga'].append(ga_fitness_curve)
    function_calls['ga'].append(len(ga_fitness_curve))

    # MIMIC
    mimic_fitness_curve = mlrose_hiive.mimic(problem,
                                             max_attempts=100,
                                             max_iters=100,
                                             curve=True,
                                             random_state=42,
                                             keep_pct=0.5)[2]
    fitness_curves['mimic'].append(mimic_fitness_curve)
    function_calls['mimic'].append(len(mimic_fitness_curve))

# Plot fitness curves for each algorithm and problem size
for algo, curves in fitness_curves.items():
    plt.figure(figsize=(10, 6))
    for i, curve in enumerate(curves):
        plt.plot(curve[:,0], label=f'Problem size: {problem_sizes[i]}')
    plt.xlabel('Iterations')
    plt.ylabel('Fitness')
    plt.title(f'Fitness Curve for {algo.upper()}')
    plt.legend()
    plt.grid(True)
    plt.show()

for algo, calls in function_calls.items():
    plt.figure(figsize=(10, 6))
    plt.plot(calls, marker='o')
    plt.xlabel('Iterations')
    plt.ylabel('Function Calls')
    plt.title(f'Function Calls vs Problem Size for {algo.upper()}')
    plt.grid(True)
    plt.show()
```



One Max

In [31]:

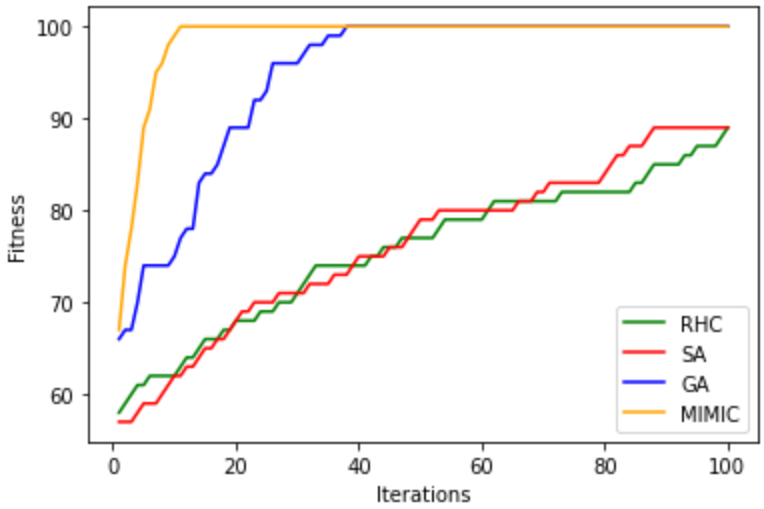
```
print("Running Experiments for One Max Problem")
print()
np.random.seed(0)

# Define Fitness function and discrete problem object
fitness = mlrose_hiive.OneMax()
problem = mlrose_hiive.DiscreteOpt(length=100, fitness_fn=fitness, maximiz

max_attempts = 100
max_iters = 100

# RHC
print("Running Random Hill Climb Experiment")
start_time = time.time()
rhc_best_state, rhc_best_fitness, rhc_fitness_curve = mlrose_hiive.random_




end_time = time.time()
rhc_time = end_time - start_time
print("Time (s): {}".format(rhc_time))
print()

# SA
print("Running Simulated Annealing Experiment")
start_time = time.time()
sa_best_state, sa_best_fitness, sa_fitness_curve = mlrose_hiive.simulated_
                                                    problem,
                                                    max_attempts=max_atter
                                                    max_iters=max_iters,
                                                    curve=True,
                                                    random_state=42,
                                                    schedule=mlrose_hiive.
end_time = time.time()
sa_time = end_time - start_time
print("Time (s): {}".format(sa_time))
print()

# GA
print("Running Genetic Algorithm Experiment")
start_time = time.time()
ga_best_state, ga_best_fitness, ga_fitness_curve = mlrose_hiive.genetic_al
                                                    problem,
                                                    max_attempts=max_atter
                                                    max_iters=max_iters,
                                                    curve=True,
                                                    random_state=42,
                                                    pop_size=100,
                                                    mutation_prob=0.2)
end_time = time.time()
ga_time = end_time - start_time
print("Time (s): {}".format(ga_time))
print()

# MIMIC
```

```
print("Running MIMIC Algorithm Experiment")
start_time = time.time()
mimic_best_state, mimic_best_fitness, mimic_fitness_curve = mlrose_hiive.r
                                                            problem,
                                                            max_attempts =
                                                            max_iters = 10
                                                            curve = True,
                                                            random_state =
                                                            keep_pct=0.25)
end_time = time.time()
mimic_time = end_time - start_time
print("Time (s): {}".format(mimic_time))
print()
```

```
Running Experiments for One Max Problem

Running Random Hill Climb Experiment
Time (s): 0.6760413646697998

Running Simulated Annealing Experiment
Time (s): 0.016124486923217773

Running Genetic Algorithm Experiment
Time (s): 1.6141231060028076

Running MIMIC Algorithm Experiment
Time (s): 1169.8148610591888
```

In [32]:
```
# Plot Iterations vs Fitness
iterations = range(1, max_iters + 1)
plt.plot(iterations, rhc_fitness_curve[:, 0], label='RHC', color='green')
plt.plot(iterations, sa_fitness_curve[:, 0], label='SA', color='red')
plt.plot(iterations, ga_fitness_curve[:, 0], label='GA', color='blue')
plt.plot(iterations, mimic_fitness_curve[:, 0], label='MIMIC', color='oran
plt.legend(loc="best")
plt.xlabel("Iterations")
plt.ylabel("Fitness")
```

Out[32]:
```
Text(0, 0.5, 'Fitness')
```



In [33]:
```
# Plot Time Table
data = [('RHC', round(rhc_time, 5)),
        ('SA', round(sa_time, 5)),
        ('GA', round(ga_time, 5)),
        ('MIMIC', round(mimic_time, 5))]

time_OneMax = pd.DataFrame(data, columns =['Algorithm', 'Time (s)'])
time_OneMax
```

Out[33]:

	Algorithm	Time (s)
0	RHC	0.67604
1	SA	0.01612
2	GA	1.61412
3	MIMIC	1169.81486

In [34]:

```
np.random.seed(0)


# Define a range of problem sizes (lengths)
problem_sizes = [10, 50, 100]

# Lists to store fitness curves for each algorithm and problem size
fitness_curves = {
    'rhc': [],
    'sa': [],
    'ga': [],
    'mimic': []
}

function_calls = {
    'rhc': [],
    'sa': [],
    'ga': [],
    'mimic': []
}

# Run experiments for each problem size
for length in problem_sizes:
    print("Running experiments for problem size: {}".format(length))

    # Define Fitness function and discrete problem object for the current
    fitness = mlrose_hiive.OneMax()
    problem = mlrose_hiive.DiscreteOpt(length=length, fitness_fn=fitness,

    # RHC
    rhc_fitness_curve = mlrose_hiive.random_hill_climb(problem,
                                                       max_attempts=max_at
                                                       max_iters=max_iters
                                                       curve=True,
                                                       random_state=42,
                                                       restarts=100)[2]
    fitness_curves['rhc'].append(rhc_fitness_curve)
    function_calls['rhc'].append(len(rhc_fitness_curve))


    # SA
    sa_fitness_curve = mlrose_hiive.simulated_annealing(problem,
                                                        max_attempts=max_at
                                                        max_iters=max_iters
                                                        curve=True,
                                                        random_state=42,
                                                        schedule=mlrose_hii
    fitness_curves['sa'].append(sa_fitness_curve)
    function_calls['sa'].append(len(sa_fitness_curve))

    # GA
    ga_fitness_curve = mlrose_hiive.genetic_alg(problem,
                                                max_attempts=max_attempts,
                                                max_iters=max_iters,
                                                curve=True,
                                                random_state=42,
                                                pop_size=100,
```

```
                                                    mutation_prob=0.2)[2]
    fitness_curves['ga'].append(ga_fitness_curve)
    function_calls['ga'].append(len(ga_fitness_curve))

    # MIMIC
    mimic_fitness_curve = mlrose_hiive.mimic(problem,
                                             max_attempts=100,
                                             max_iters=100,
                                             curve=True,
                                             random_state=42,
                                             keep_pct=0.25)[2]
    fitness_curves['mimic'].append(mimic_fitness_curve)
    function_calls['mimic'].append(len(mimic_fitness_curve))

# Plot fitness curves for each algorithm and problem size
for algo, curves in fitness_curves.items():
    plt.figure(figsize=(10, 6))
    for i, curve in enumerate(curves):
        plt.plot(curve[:,0], label=f'Problem size: {problem_sizes[i]}')
    plt.xlabel('Iterations')
    plt.ylabel('Fitness')
    plt.title(f'Fitness Curve for {algo.upper()}')
    plt.legend()
    plt.grid(True)
    plt.show()

for algo, calls in function_calls.items():
    plt.figure(figsize=(10, 6))
    plt.plot(calls, marker='o')
    plt.xlabel('Iterations')
    plt.ylabel('Function Calls')
    plt.title(f'Function Calls vs Problem Size for {algo.upper()}')
    plt.grid(True)
    plt.show()
```

```
Running experiments for problem size: 10
Running experiments for problem size: 50
Running experiments for problem size: 100
```