experiments

October 24, 2023

[22]:
```python
import os
import time
import numpy as np
import matplotlib.pyplot as plt

from simulated_annealing import SimulatedAnnealing
from evolutionary_search import EvolutionarySearch
from stochastic_beam_search import StochasticBeamSearch

%matplotlib qt
```

[2]:
```python
# csv file path
csv_path = os.path.join(os.getcwd(), "hw2.csv")
```

[20]:
```python
## Simulated Annealing experiments
# Parameters
iterations = 5000
temperature = 15.0
temperature_decay = 0.9
num_swaps = 10
runs = 20

# Results
best_paths = []
best_cost = []
iteration_costs = []
time_to_run = []

# Run simulated annealing for number of runs
for i in range(runs):
    # Initialize simulated annealing
    sa = SimulatedAnnealing(csv_path, iterations, temperature,␣
 ↪temperature_decay, num_swaps)

    # Run simulated annealing
    start = time.time()
    sa.algorithm(verbose=False)
```

```
    end = time.time()

    # Save results
    best_paths.append(sa.best_path)
    best_cost.append(sa.best_cost)
    iteration_costs.append(sa.cost_history)
    time_to_run.append(end - start)
```

[35]:

```
# Find best run
best_run_idx = np.argmin(best_cost)

# Plot iteration costs for best run
plt.figure("Iteration costs for best run")
plt.plot(range(len(iteration_costs[best_run_idx])),
  ↪iteration_costs[best_run_idx])
plt.xlabel("Iteration")
plt.ylabel("Cost")
plt.title("Iteration Costs for Best run")

# Plot best path for best run
sa._plot_path(best_paths[best_run_idx])

# Plot best cost for each run
plt.figure("Best cost for each run")
plt.scatter(range(1, runs+1), best_cost, s=20)
plt.xticks(range(1, runs+1))
plt.xlabel("Run")
plt.ylabel("Best Cost")
plt.title("Best Cost for each run")

# Plot iteration costs for each run
plt.figure("Iteration costs for each run")
for i in range(runs):
    plt.plot(range(len(iteration_costs[i])), iteration_costs[i])

plt.xlabel("Iteration")
plt.ylabel("Cost")
plt.title("Iteration Costs for all runs")

plt.show()
```

[38]:

```
print("For Simulated Annealing:")

# Print the time to run for each run
print(f"Time taken for each run in sec: {time_to_run}")

# Print the best cost achieved
```

```python
print(f"Best cost achieved: {best_cost[best_run_idx]}")

# Print the mean and standard deviation of the best cost
print(f"Mean best cost: {np.mean(best_cost)}")
print(f"Standard deviation of best cost: {np.std(best_cost)}")
```

```
For Simulated Annealing:
Time taken for each run in sec: [0.27204298973083496, 0.22841525077819824,
0.22993993759155273, 0.2314624786376953, 0.22998046875, 0.22643208503723145,
0.22549843788146973, 0.2241065502166748, 0.22742605209350586,
0.22686243057250977, 0.22619199752807617, 0.22486424446105957,
0.22822117805480957, 0.2328319549560547, 0.2298872470855713,
0.23218560218811035, 0.22817301750183105, 0.2301175594329834,
0.2245655059814453, 0.22311997413635254]
Best cost achieved: 4.331013482064009
Mean best cost: 4.7527596110478045
Standard deviation of best cost: 0.2761799610804234
```

[40]:
```python
## Evolutionary Search experiments
# Parameters
population_size = 100
iterations = 1000
num_swaps = 10
mutation_size = 90
runs = 20

# Results
best_paths = []
best_cost = []
iteration_costs = []
time_to_run = []

# Run evolutionary search for number of runs
for i in range(runs):
    # Initialize evolutionary search
    es = EvolutionarySearch(csv_path, population_size, iterations, num_swaps,␣
 ↪mutation_size)

    # Run evolutionary search
    start = time.time()
    es.algorithm(verbose=False)
    end = time.time()

    # Save results
    best_paths.append(es.best_path)
    best_cost.append(es.best_cost)
    iteration_costs.append(es.generational_cost)
```

```
    time_to_run.append(end - start)
```

[42]:
```
# Find best run
best_run_idx = np.argmin(best_cost)

# Plot generation costs for best run
plt.figure("Generation costs for best run")
for i in range(len(iteration_costs[best_run_idx])):
    plt.scatter(np.repeat(i, population_size),␣
 ↪iteration_costs[best_run_idx][i], s=1)
plt.xlabel("Generation")
plt.ylabel("Cost")
plt.title("All Generation Costs for Best run")

# Plot best costs in each generation for the best run
plt.figure("Best costs for best run")
plt.plot(range(len(iteration_costs[best_run_idx])), np.
 ↪min(iteration_costs[best_run_idx], axis=1))
plt.xlabel("Generation")
plt.ylabel("Cost")
plt.title("Best Costs over generation for Best run")

# Plot best path for best run
es._plot_path(best_paths[best_run_idx])

# Plot best cost for each run
plt.figure("Best cost for each run")
plt.scatter(range(1, runs+1), best_cost, s=20)
plt.xticks(range(1, runs+1))
plt.xlabel("Run")
plt.ylabel("Best Cost")
plt.title("Best Cost for each run")

# Plot generational best costs for each run
plt.figure("Generational best costs for each run")
for i in range(runs):
    plt.plot(range(len(iteration_costs[i])), np.min(iteration_costs[i], axis=1))
plt.xlabel("Generation")
plt.ylabel("Cost")
plt.title("Generational Best Costs for all runs")

plt.show()
```

[43]:
```
print("For Evolutionary Search:")

# Print the time to run for each run
print(f"Time taken for each run in sec: {time_to_run}")
```

```
# Print the best cost achieved
print(f"Best cost achieved: {best_cost[best_run_idx]}")

# Print the mean and standard deviation of the best cost
print(f"Mean best cost: {np.mean(best_cost)}")
print(f"Standard deviation of best cost: {np.std(best_cost)}")
```

```
For Evolutionary Search:
Time taken for each run in sec: [3.8853065967559814, 3.604876756668091,
2.3146116733551025, 2.3303232192993164, 2.3131256103515625, 2.3504374027252197,
2.3750338554382324, 2.2965402603149414, 2.3313534259796143, 2.3383419513702393,
2.3685531616210938, 2.3826637268066406, 2.356966972351074, 2.4064712524414062,
2.37369441986084, 2.38724422454834, 2.372828960418701, 2.3999781608581543,
2.3677496910095215, 2.386867046356201]
Best cost achieved: 3.9708135314285755
Mean best cost: 4.2460881766863166
Standard deviation of best cost: 0.16885564554528337
```

[45]:

```
## Stochastic Beam Search experiments
# Parameters
beam_width = 25
iterations = 1000
num_swaps = 10
stochastic_factor = 1.0
cooling_rate = 0.9
runs = 20

# Results
best_paths = []
best_cost = []
iteration_costs = []
time_to_run = []

# Run evolutionary search for number of runs
for i in range(runs):
    # Initialize Stochastic Beam Search
    sbs = StochasticBeamSearch(csv_path, beam_width, iterations, num_swaps,␣
 ↪stochastic_factor, cooling_rate)

    # Run evolutionary search
    start = time.time()
    sbs.algorithm(verbose=False)
    end = time.time()

    # Save results
    best_paths.append(sbs.best_path)
```

```python
    best_cost.append(sbs.best_cost)
    iteration_costs.append(sbs.iterational_cost)
    time_to_run.append(end - start)
```

[46]:
```python
# Find best run
best_run_idx = np.argmin(best_cost)

# Plot iteration costs for best run
plt.figure("Iteration costs for best run")
for i in range(len(iteration_costs[best_run_idx])):
    plt.scatter(np.repeat(i, beam_width), iteration_costs[best_run_idx][i], s=1)
plt.xlabel("Iteration")
plt.ylabel("Cost")
plt.title("All Iteration Costs for Best run")

# Plot best costs in each iteration for the best run
plt.figure("Best costs for best run")
plt.plot(range(len(iteration_costs[best_run_idx])), np.
 ↪min(iteration_costs[best_run_idx], axis=1))
plt.xlabel("Iteration")
plt.ylabel("Cost")
plt.title("Best Costs over iteration for Best run")

# Plot best path for best run
sbs._plot_path(best_paths[best_run_idx])

# Plot best cost for each run
plt.figure("Best cost for each run")
plt.scatter(range(1, runs+1), best_cost, s=20)
plt.xticks(range(1, runs+1))
plt.xlabel("Run")
plt.ylabel("Best Cost")
plt.title("Best Cost for each run")

# Plot iterational best costs for each run
plt.figure("Iterational best costs for each run")
for i in range(runs):
    plt.plot(range(len(iteration_costs[i])), np.min(iteration_costs[i], axis=1))
plt.xlabel("Iteration")
plt.ylabel("Cost")
plt.title("Iterational Best Costs for all runs")

plt.show()
```

[47]:
```python
print("For Stochastic Beam Search:")

# Print the time to run for each run
```

```python
print(f"Time taken for each run in sec: {time_to_run}")

# Print the best cost achieved
print(f"Best cost achieved: {best_cost[best_run_idx]}")

# Print the mean and standard deviation of the best cost
print(f"Mean best cost: {np.mean(best_cost)}")
print(f"Standard deviation of best cost: {np.std(best_cost)}")
```

```
For Stochastic Beam Search:
Time taken for each run in sec: [1.2976090908050537, 1.2390804290771484,
1.2067830562591553, 1.2253077030181885, 1.2317626476287842, 1.1565864086151123,
0.8249421119689941, 0.8102307319641113, 0.7752177715301514, 0.759474515914917,
0.7691755294799805, 0.7858595848083496, 0.7851295471191406, 0.7573051452636719,
0.7707412242889404, 0.7716658115386963, 0.8011846542358398, 0.7934019565582275,
0.7988109588623047, 0.787672758102417]
Best cost achieved: 4.058206062763929
Mean best cost: 4.440742810815573
Standard deviation of best cost: 0.24586903082127312
```