TSP Analysis

October 6, 2025

1 Traveling Salesman Problem: Algorithm Comparison

This notebook compares six TSP algorithms on the Lin105 dataset with a 5-second time limit: Random Solver, Nearest Neighbor; Simulated Annealing and Genetic Algorithm (with random and NN initialization).

We compare them by looking at their final cost versus the optimal and baseline, how their cost changes over time, how many steps they take per second, and how much they improve from the starting point.

[21]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
from tsp.model import TSPInstance
from algorithm.nearest_neighbor import NearestNeighbor
from algorithm.simulated_annealing import SimulatedAnnealing
from algorithm.genetic_algo import GeneticAlgorithmSolver
from algorithm.random_solver import RandomSolver
from util import exponential_cooling, find_optimal_tour,␣
↪run_algorithm_with_timing
```

[ ]:
```
# Config
MAX_SECONDS = 5.0

# SA/GA parameters (from tuning run)
T0 = 167.807
COOLING_RATE = 0.99990
GA_POP_SIZE = 109
GA_CROSSOVER = 0.600
GA_MUTATION = 0.400
GA_ELITISM = 1

# Cooling schedule
exp_schedule = exponential_cooling(COOLING_RATE)

# Algorithm lineup
ALGORITHMS = [
```

```
    "Random Solver",
    "Nearest Neighbor",
    "SA (Random-init)",
    "SA (NN-init)",
    "GA (Random-init)",
    "GA (NN-init)",
]

# Plot styling
plt.rcParams['figure.figsize'] = (10, 6)
plt.rcParams['font.size'] = 11
```

[24]:
```
problem_instance_path = Path("dataset/lin105.tsp")
instance, optimal_cost = find_optimal_tour(problem_instance_path)
```

[ ]:
```
# Utils
def get_nn_route_and_cost(instance):
    builder = NearestNeighbor(instance)
    builder.initialize(None)
    for _ in range(len(instance.cities) - 1):
        builder.step()
    return builder.get_route(), builder.get_cost()

def run_single_time_trial(name, instance_data, seed_nn_data):
    inst = TSPInstance(name=instance_data["name"],␣
 ↪cities=instance_data["cities"])

    if name == "Random Solver":
        solver, init_route = RandomSolver(inst), None
    elif name == "Nearest Neighbor":
        solver, init_route = NearestNeighbor(inst), None
    elif name == "SA (Random-init)":
        solver, init_route = SimulatedAnnealing(inst, T0, exp_schedule), None
    elif name == "SA (NN-init)":
        solver, init_route = SimulatedAnnealing(inst, T0, exp_schedule),␣
 ↪seed_nn_data
    elif name == "GA (Random-init)":
        solver = GeneticAlgorithmSolver(
            inst,
            population_size=GA_POP_SIZE,
            mutation_rate=GA_MUTATION,
            crossover_rate=GA_CROSSOVER,
            elitism_count=GA_ELITISM,
        )
        init_route = None
    elif name == "GA (NN-init)":
        solver = GeneticAlgorithmSolver(
```

```
            inst,
            population_size=GA_POP_SIZE,
            mutation_rate=GA_MUTATION,
            crossover_rate=GA_CROSSOVER,
            elitism_count=GA_ELITISM,
        )
        init_route = seed_nn_data
    else:
        raise ValueError(f"Unknown algorithm name: {name}")

    iters, best, curr, times, route = run_algorithm_with_timing(inst, solver,␣
↪init_route, MAX_SECONDS)
    steps_per_sec = (len(iters) / times[-1]) if times else 0.0
    return {
        "name": name,
        "iterations": iters,
        "best_costs": best,
        "current_costs": curr,
        "times": times,
        "route": route,
        "final_cost": best[-1] if best else float("inf"),
        "steps_per_sec": steps_per_sec,
    }
```

[40]:

```
seed_nn, nn_cost = get_nn_route_and_cost(instance)
```

[ ]:

```
# Run algorithms (for 30 secs total = n_algos * MAX_SECONDS)
instance_data = {"name": instance.name, "cities": instance.cities}

time_runs = {name: [] for name in ALGORITHMS}
for name in ALGORITHMS:
    time_runs[name] = [run_single_time_trial(name, instance_data, seed_nn)]

time_results = {}
for name, runs in time_runs.items():
    r = runs[0] if runs else {}
    x = r.get("times", [])
    y = r.get("best_costs", [])
    time_results[name] = {
        "times": x,
        "best": np.array(y, dtype=float),
        "final_cost": float(r.get("final_cost", float('inf'))),
        "steps_per_sec": float(r.get("steps_per_sec", 0.0)),
    }
```

[73]:

```
summary_rows = []
for name, data in time_results.items():
```

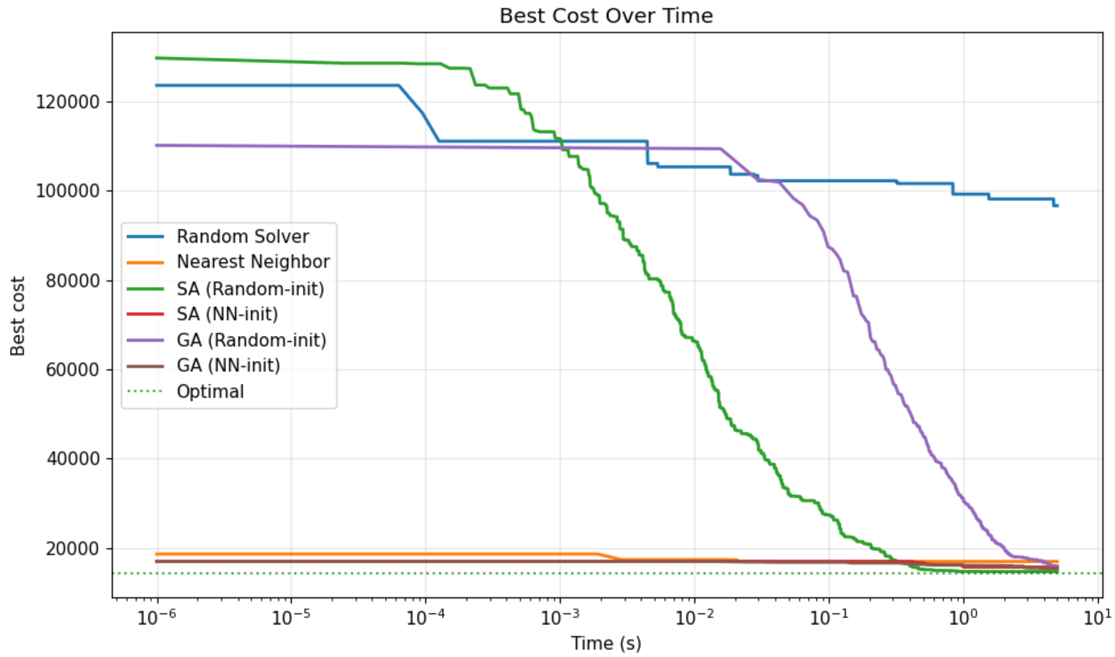
```
    cost = data["final_cost"]
    sps = data["steps_per_sec"]
    summary_rows.append({
        "Algorithm": name,
        "Final Cost": f"{cost:.1f}",
        "Steps/sec": f"{sps:.1f}",
    })

summary_rows.append({
    "Algorithm": "Optimal (ref)",
    "Final Cost": f"{optimal_cost:.1f}",
    "Steps/sec": "",
})

pd.DataFrame(summary_rows).sort_values("Final Cost")
```

[73]:

	Algorithm	Final Cost	Steps/sec
6	Optimal (ref)	14383.0	
2	SA (Random-init)	14677.2	45987.0
5	GA (NN-init)	15390.3	139.5
3	SA (NN-init)	15692.4	46168.0
4	GA (Random-init)	15958.7	135.6
1	Nearest Neighbor	16939.4	988.0
0	Random Solver	96658.0	29824.4

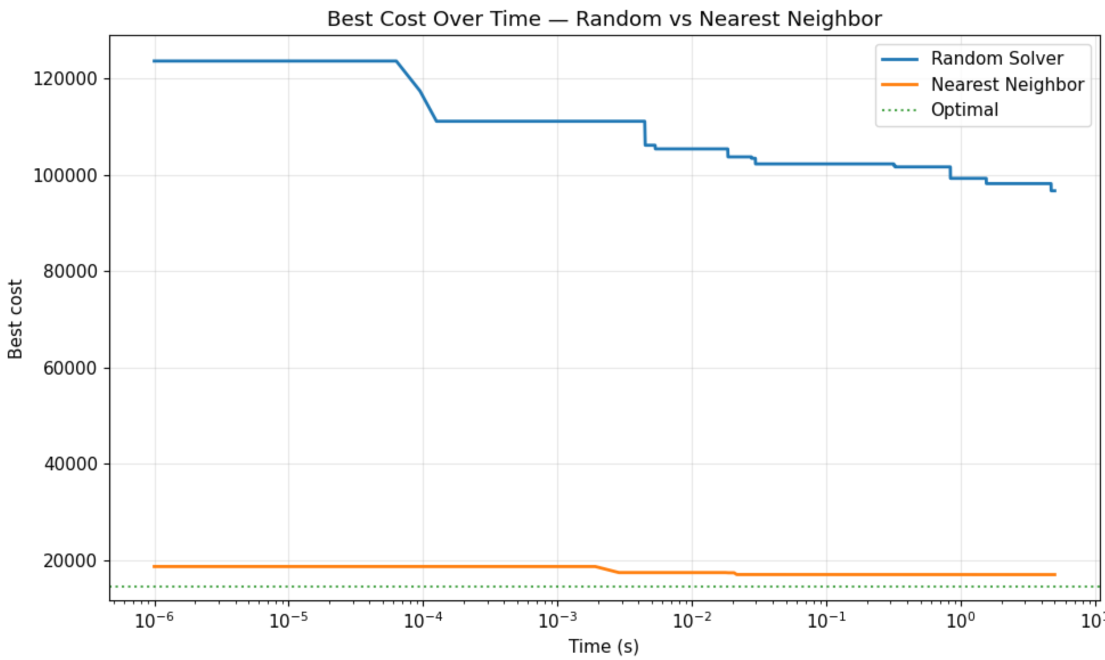
[ ]:

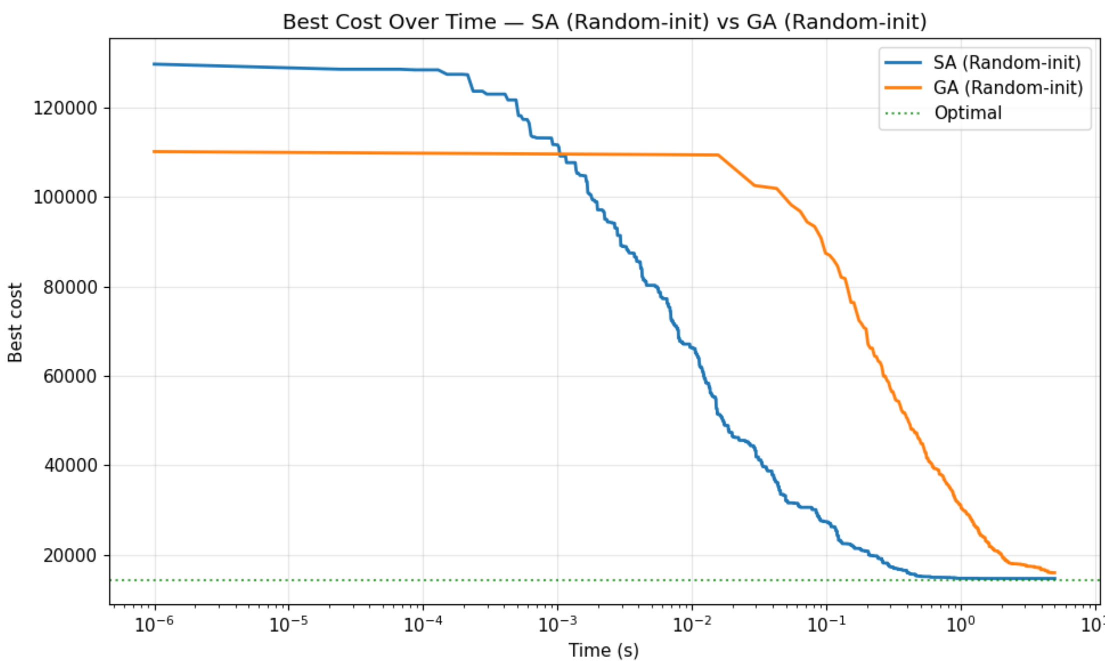
```
fig, ax = plt.subplots()
for name, data in time_results.items():
    ax.plot(data["times"], data["best"], label=name, linewidth=2)
ax.axhline(y=optimal_cost, color="green", linestyle=":", label="Optimal",␣
 ↪alpha=0.7)
ax.set_xlabel("Time (s)")
ax.set_ylabel("Best cost")
ax.set_title('Best Cost Over Time')
ax.set_xscale('log')
ax.grid(True, alpha=0.3)
ax.legend()
plt.tight_layout()
plt.show()
```

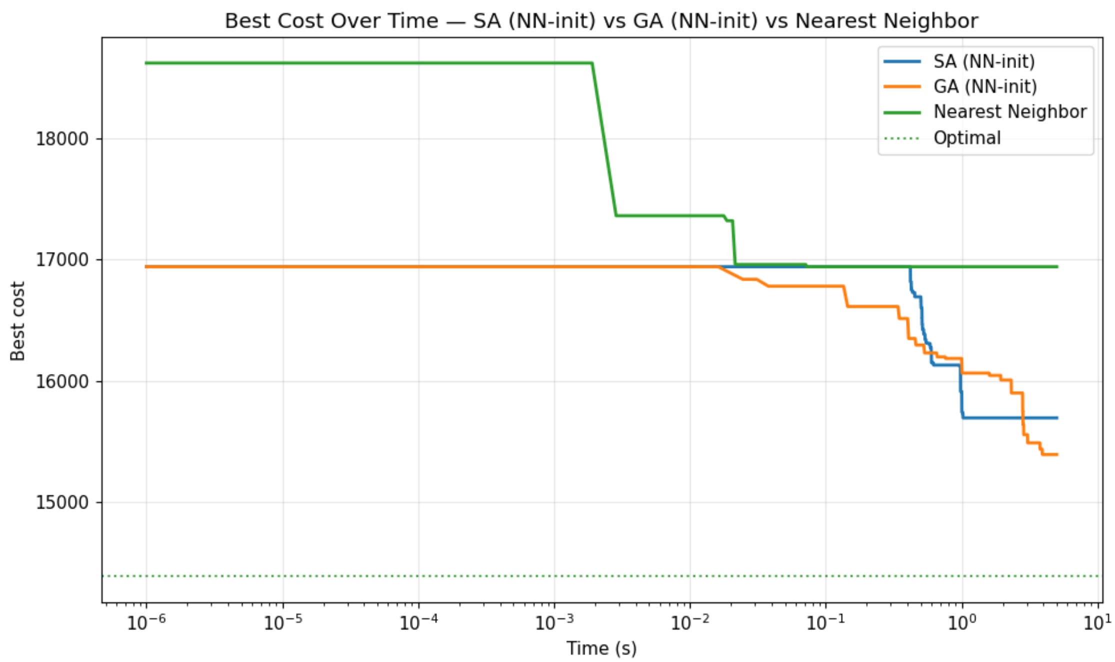


[ ]:
```python
def plot_best_over_time_for(names_subset, title):
    fig, ax = plt.subplots()
    for n in names_subset:
        if n in time_results:
            ax.plot(time_results[n]["times"], time_results[n]["best"], label=n,␣
 ↪linewidth=2)
    if optimal_cost:
        ax.axhline(y=optimal_cost, color="green", linestyle=":",␣
 ↪label="Optimal", alpha=0.7)
    ax.set_xlabel("Time (s)")
    ax.set_ylabel("Best cost")
    ax.set_title(title)
    ax.set_xscale('log')
    ax.grid(True, alpha=0.3)
    ax.legend()
    plt.tight_layout()
    plt.show()

plot_best_over_time_for([
    "Random Solver", "Nearest Neighbor"
], "Best Cost Over Time - Random vs Nearest Neighbor")
plot_best_over_time_for([
    "SA (Random-init)", "GA (Random-init)"
], "Best Cost Over Time - SA (Random-init) vs GA (Random-init)")
plot_best_over_time_for([
```

```
    "SA (NN-init)", "GA (NN-init)", "Nearest Neighbor"
], "Best Cost Over Time - SA (NN-init) vs GA (NN-init) vs Nearest Neighbor")
```

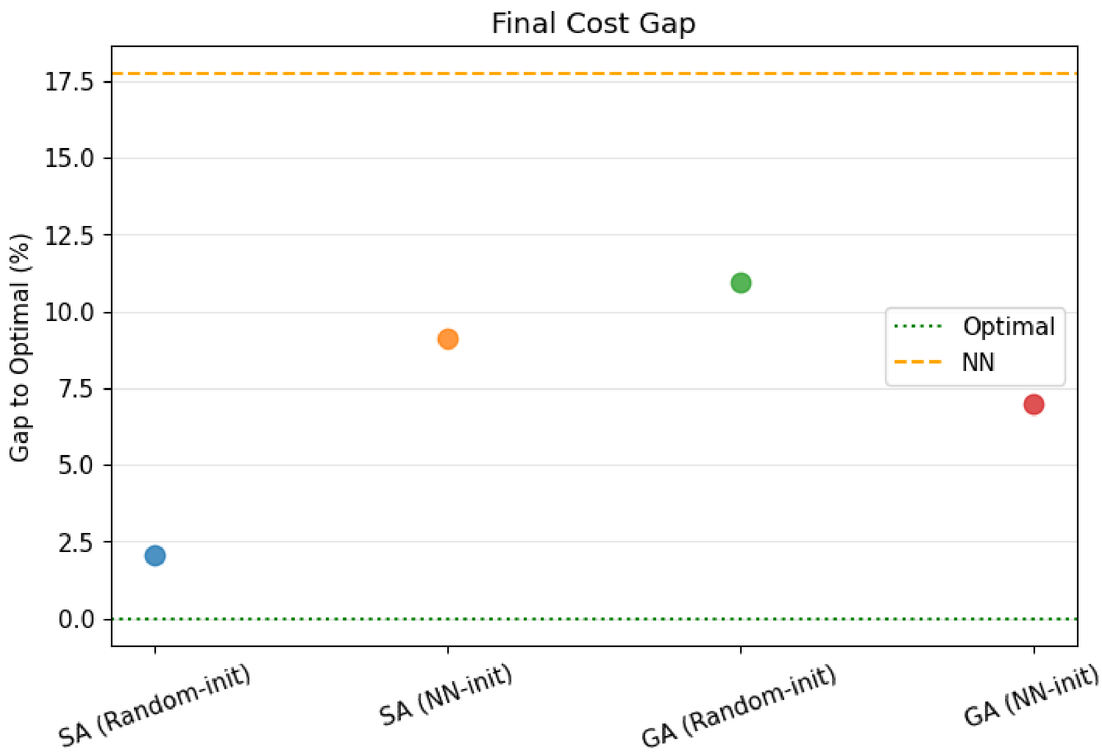






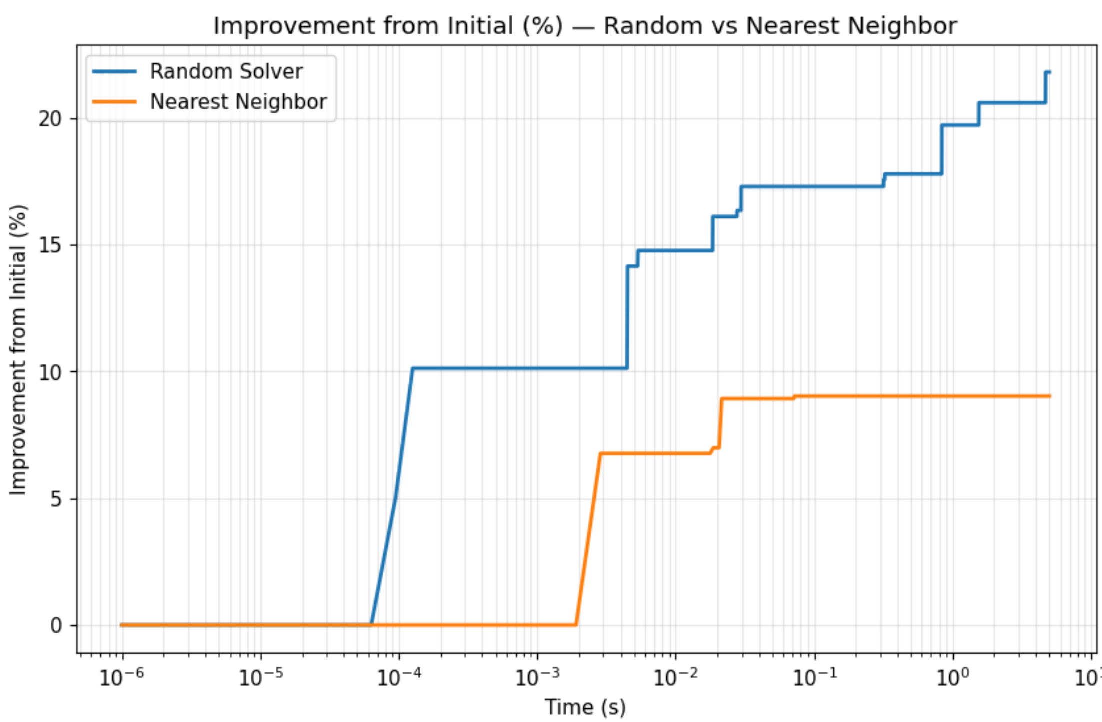
[75]:
```
names = [n for n in time_runs.keys() if (not n.lower().startswith('random')) ␣
↪and (not n.lower().startswith('nearest'))]

final_costs_by_algo = {n: [] for n in names}
for n in names:
    runs = time_runs.get(n, [])
    for r in runs:
        best_costs = r.get("best_costs", [])
        if best_costs:
            final_costs_by_algo[n].append(best_costs[-1])

gaps_by_algo = {}
for n, costs in final_costs_by_algo.items():
    gaps = [((c / optimal_cost) - 1) * 100.0 for c in costs if c > 0]
    gaps_by_algo[n] = gaps

fig, ax = plt.subplots(figsize=(8, 5))

for i, n in enumerate(names):
    y = gaps_by_algo[n]
    x = np.full(len(y), i)
    ax.scatter(x, y, s=80, label=n, alpha=0.8)

ax.set_xticks(range(len(names)))
ax.set_xticklabels(names, rotation=20)
```

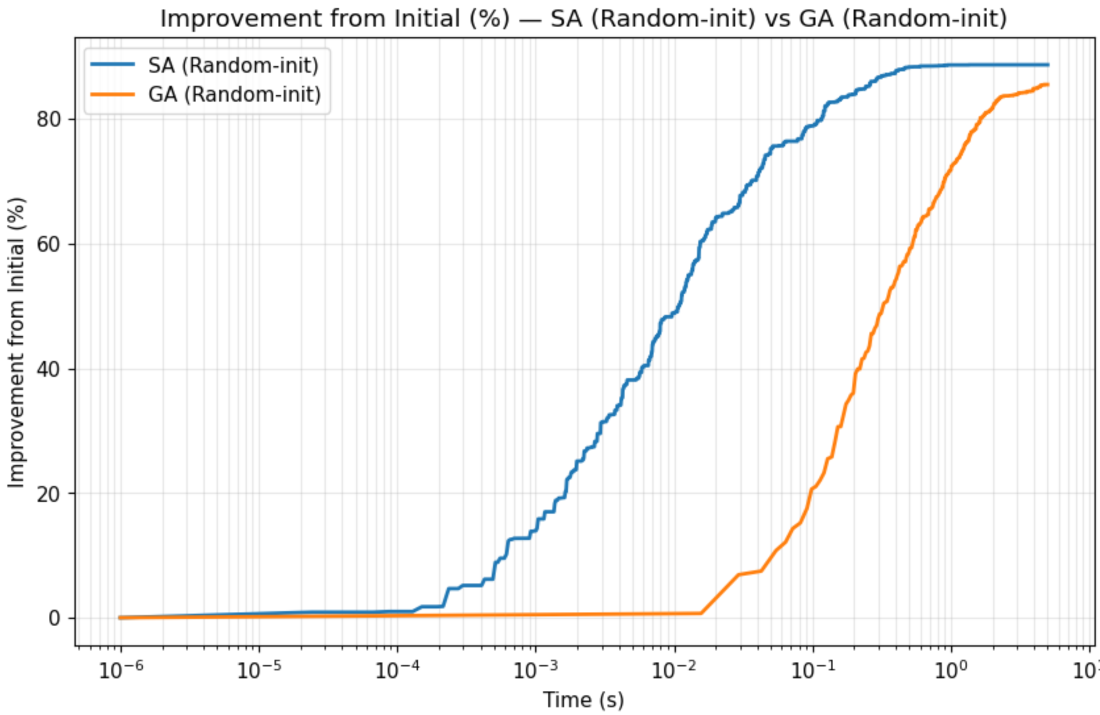
```
ax.set_ylabel('Gap to Optimal (%)')
ax.set_title('Final Cost Gap')
ax.grid(True, axis='y', alpha=0.3)

nn_gap = ((nn_cost / optimal_cost) - 1) * 100.0
opt_line = ax.axhline(y=0, color='green', linestyle=':', label='Optimal')
nn_line = ax.axhline(y=nn_gap, color='orange', linestyle='--', label='NN')
ax.legend(handles=[opt_line, nn_line])
plt.show()
```



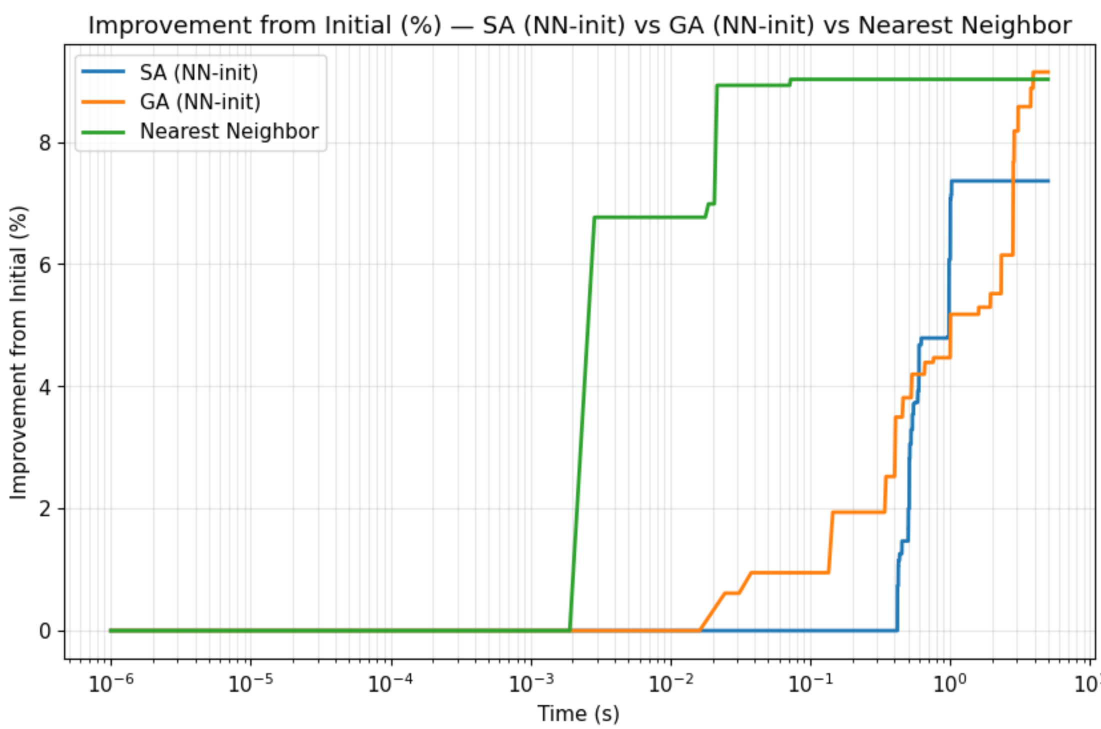

```
[ ]: def plot_relative_improvement_over_time(names_subset, title=None):
         fig, ax = plt.subplots()
         for n in names_subset:
             if n in time_results:
                 times = time_results[n]['times']
                 best = time_results[n]['best']
                 if len(best) > 0 and best[0] > 0:
                     initial = float(best[0])
                     improvement_pct = (initial - best) / initial * 100.0
                     ax.plot(times, improvement_pct, label=n, linewidth=2)
         ax.set_xlabel('Time (s)')
         ax.set_ylabel('Improvement from Initial (%)')
```

```python
    ax.set_title(title)
    ax.set_xscale('log')
    ax.legend()
    ax.grid(True, which='both', axis='x', alpha=0.3)
    ax.grid(True, which='major', axis='y', alpha=0.3)
    plt.show()

plot_relative_improvement_over_time(
    ["Random Solver", "Nearest Neighbor"],
    "Improvement from Initial (%) - Random vs Nearest Neighbor"
)
plot_relative_improvement_over_time(
    ["SA (Random-init)", "GA (Random-init)"],
    "Improvement from Initial (%) - SA (Random-init) vs GA (Random-init)"
)
plot_relative_improvement_over_time(
    ["SA (NN-init)", "GA (NN-init)", "Nearest Neighbor"],
    "Improvement from Initial (%) - SA (NN-init) vs GA (NN-init) vs Nearest␣
↪Neighbor"
)
```





Improvement from Initial (%) — SA (Random-init) vs GA (Random-init)
SA (Random-init)
GA (Random-init)
Improvement from Initial (%)
Time (s)



Improvement from Initial (%) — SA (NN-init) vs GA (NN-init) vs Nearest Neighbor
SA (NN-init)
GA (NN-init)
Nearest Neighbor
Improvement from Initial (%)
Time (s)