lab

April 26, 2022

0.1 # Metaheuristics Framework

0.1.1 Notebook 1 - Simple Metaheuristics

0.2 Imports

[2]:
```
# Python libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Metaheuristics Framework Library
from metaheuristics.simulated_annealing import SimulatedAnnealing
from metaheuristics.tabu_search import TabuSearch
from metaheuristics.genetic_algorithm import GeneticAlgorithm
from metaheuristics.variable_neighborhood_descent import␣
 ↪VariableNeighborhoodDescent

# Problem modeled with the Metaheuristics Framework
from loading_models import load_solomon, load_data
from vrptw import VRPTW, FlexVRPTW
from flowshop import FlowShop, load_flowshop
```

0.3 Problem Instanciation

Choose a problem to work on

VRPTW

[5]:
```
vrptw_data = load_solomon('A50.csv', nb_cust=None, vehicle_speed=100)
problem = VRPTW(vrptw_data)
```

FlexibleVRPTW

[4]:
```
vrptw_data = load_solomon('A50.csv', nb_cust=None, vehicle_speed=100)
problem = FlexVRPTW(vrptw_data)
```

Machine Flow Shop Scheduling Problem

[ ]:
```
flowshop_data = load_flowshop('FS10x100.csv')
problem = FlowShop(flowshop_data)
```

[ ]:
```
problem.print_class_params()
```

0.4 Neighborhood lab for VRPTW

0.4.1 Utils for VRPTW demostration

[7]:
```
vrptw = problem
def reduce_solution(s, N, it=10):
    new_sol = s
    for i in range(it):
        new_sol = N.delete_smallest_route(new_sol)
    return new_sol

def print_different_route(s1, s2):
    s1, s2 = s1.routes, s2.routes
    if len(s1) != len(s2):
        return
    for i in range(len(s1)):
        if s1[i] != s2[i]:
            print('found differt route in index', i)
            print(s1[i])
            print(s2[i])

def hard_equal(a, b):
    return not set([tuple(l) for l in a]) ^ set([tuple(l) for l in b])
```

[10]:
```
N = vrptw.neighborhood({'verbose': 0, 'choose_mode': 'crossover'})
s1, s2 = N.initial_solution(), N.initial_solution()
s1, s2 = reduce_solution(s1, N), reduce_solution(s2, N)
s3, s4 = N.get_neighbor_from_two(s1, s2)
print('cost s1 =', s1.cost(), 'cost s2 =', s2.cost())
print("cost s1' =", s3.cost(), "cost s2' =", s4.cost())
print('s1 improved by', s1.cost() - s3.cost())
print('s2 improved by', s2.cost() - s4.cost())
print('s1 equal s3 =', s1==s3)
print('s2 equal s4 =', s2==s4)
print('is s3 valid ?', s3.checker(), '| is s4 valid ?', s4.checker())
print()
print_different_route(s1, s3)
print()
print_different_route(s2, s4)
```

```
cost s1 = 14232.174271580156 cost s2 = 15840.785124677413
cost s1' = 15323.156522053938 cost s2' = 14918.242055992308
s1 improved by -1090.9822504737822
s2 improved by 922.543068685105
s1 equal s3 = False
s2 equal s4 = False
```

```
is s3 valid ? True | is s4 valid ? True
```

[ ]:
```
N = problem.neighborhood({'verbose': 0, 'use_methods': [1,2,3,4,5,6,7,8],␣
 ↪'force_new_sol': True, 'full_search': False})
sol = N.initial_solution()
s = sol
# s = reduce_solution(s, it=10)
```

[ ]:
```
redN = problem.neighborhood()
s = reduce_solution(s, redN, it=30)
```

[ ]:
```
s
```

[ ]:
```
print(f'cost = {s.cost()}')
# print(s)
s2 = N(s)
print(f'cost = {s2.cost()}')
# print(s2)
print('solutions exactly equal:', hard_equal(s.routes, s2.routes))
print('improved by', s.cost() - s2.cost())
print_different_route(s, s2)
print('is s1 valid ?', s.checker(), '| is s2 valid ?', s2.checker())
```

[ ]:
```
# s = reduce_solution(s)
```

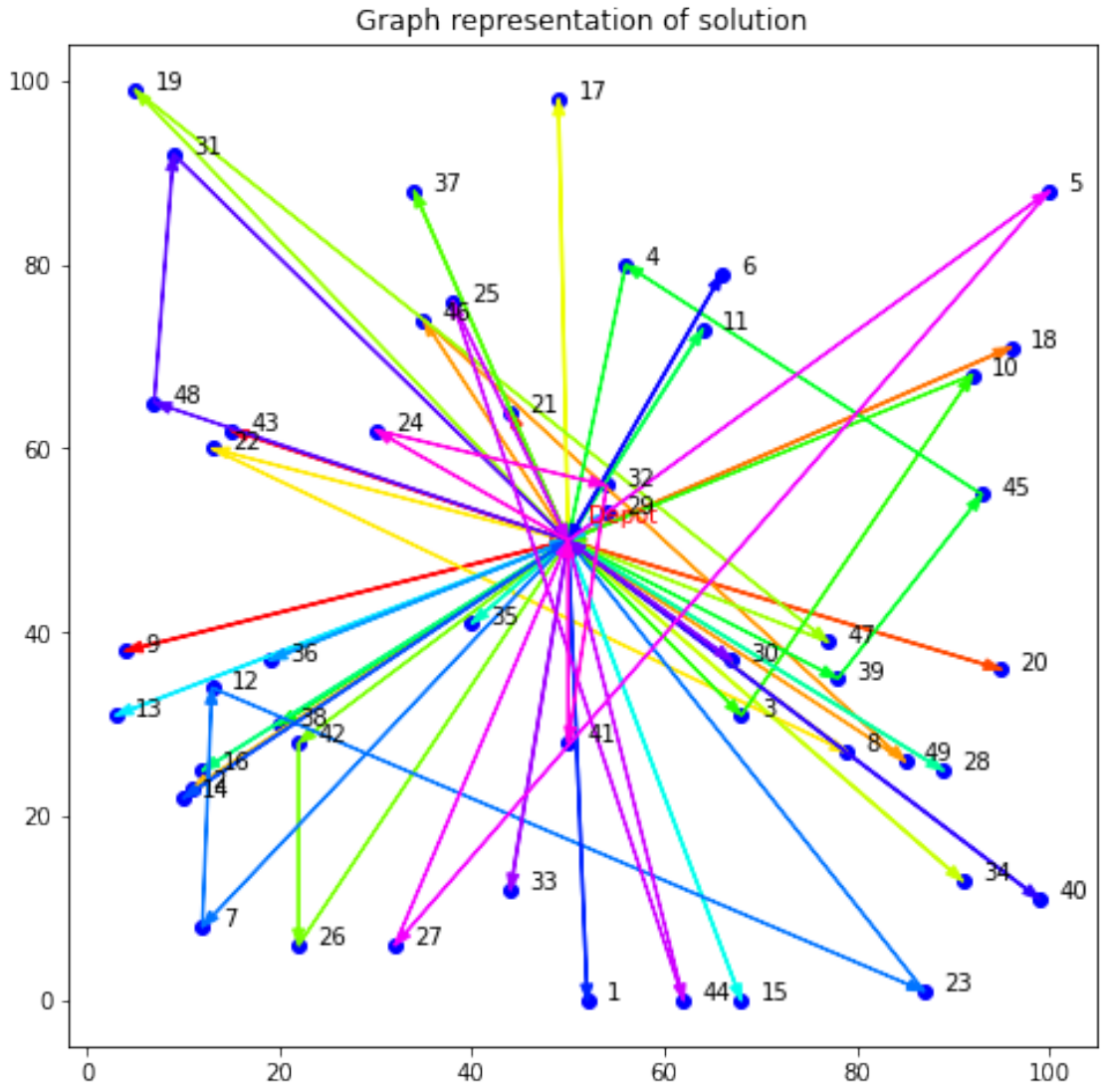
[ ]:
```
s.plot_graph(arrows=True, figsize=8)
```

1 Metaheuristics Lab

1.0.1 Initial solution for every metaheuristic

[6]:
```
N = problem.neighborhood()
init_sol = N.initial_solution()
print('cost of initial solution =', init_sol.cost())
init_sol.plot_graph(figsize=8)
```

```
cost of initial solution = 21076.79658668744
```



[7]:
```
if isinstance(problem, VRPTW) or isinstance(problem, FlexVRPTW):
    neighborhood_params={'verbose': 0,
                         'init_sol': init_sol,
                         'choose_mode': 'random',
                         'use_methods': [1, 2, 3, 4, 5, 6, 7, 8],
                         'force_new_sol': True,
                         'full_search': True}
    print('Using VRPTW neighborhood params')

elif isinstance(problem, FlowShop):
    neighborhood_params={'verbose': 0,
                         'init_sol': init_sol,
                         'choose_mode': 'random',
```

```
                        'use_methods': [1, 2],
                        'force_new_sol': True}
    print('Using Machine Flow Shop Scheduling Problem neighborhood params')
```

```
Using VRPTW neighborhood params
```

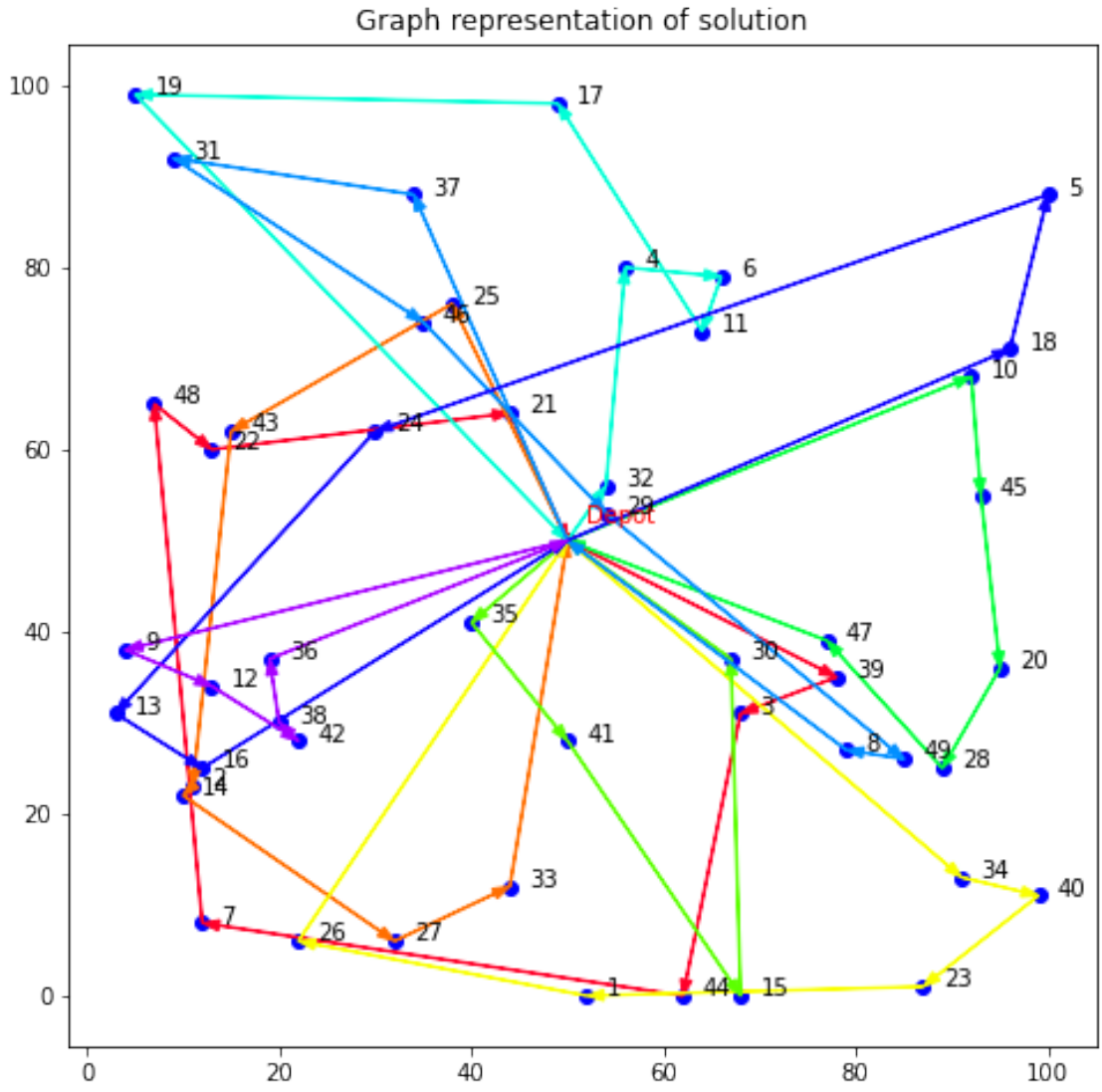
1.1 Simulated Annealing

[8]:
```
rs = SimulatedAnnealing(t0=20, progress_bar=True,␣
 ↪neighborhood_params=neighborhood_params)
rs_sol = rs.fit_search(problem)
rs_sol
```

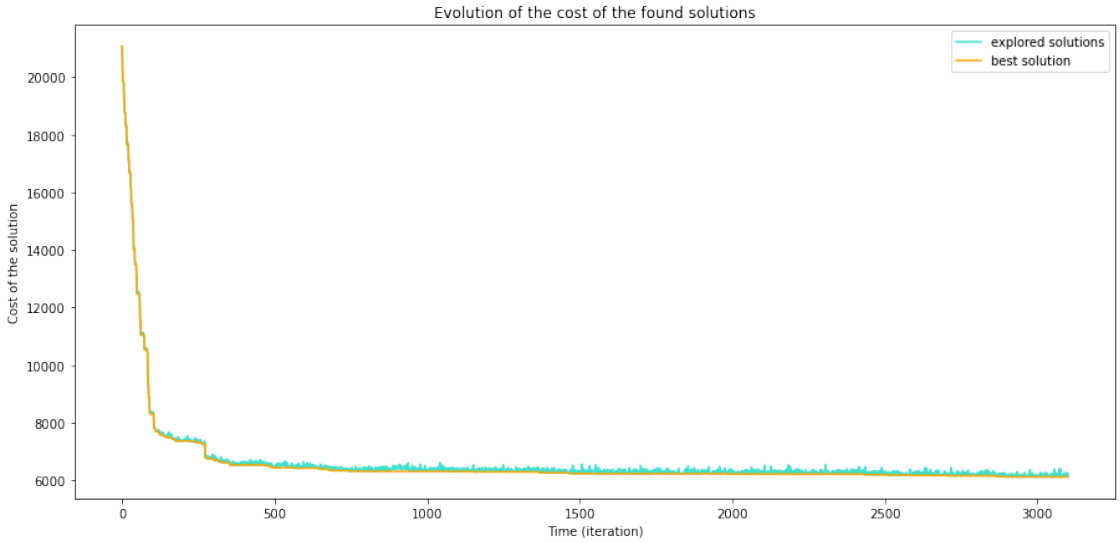
```
Cost: 6117.80:   0%|          | 0/100 [00:00<?, ?it/s]
```

[8]:
```
[[0, 39, 3, 44, 7, 48, 22, 21, 0],
 [0, 25, 43, 2, 14, 27, 33, 0],
 [0, 34, 40, 23, 1, 26, 0],
 [0, 35, 41, 15, 30, 0],
 [0, 10, 45, 20, 28, 47, 0],
 [0, 32, 4, 6, 11, 17, 19, 0],
 [0, 37, 31, 46, 29, 49, 8, 0],
 [0, 18, 5, 24, 13, 16, 0],
 [0, 9, 12, 42, 38, 36, 0]]
```

[9]:
```
rs_sol.plot_graph(figsize=8)
```



[10]:
```
rs.plot_evolution_cost(figsize=(15,7))
```



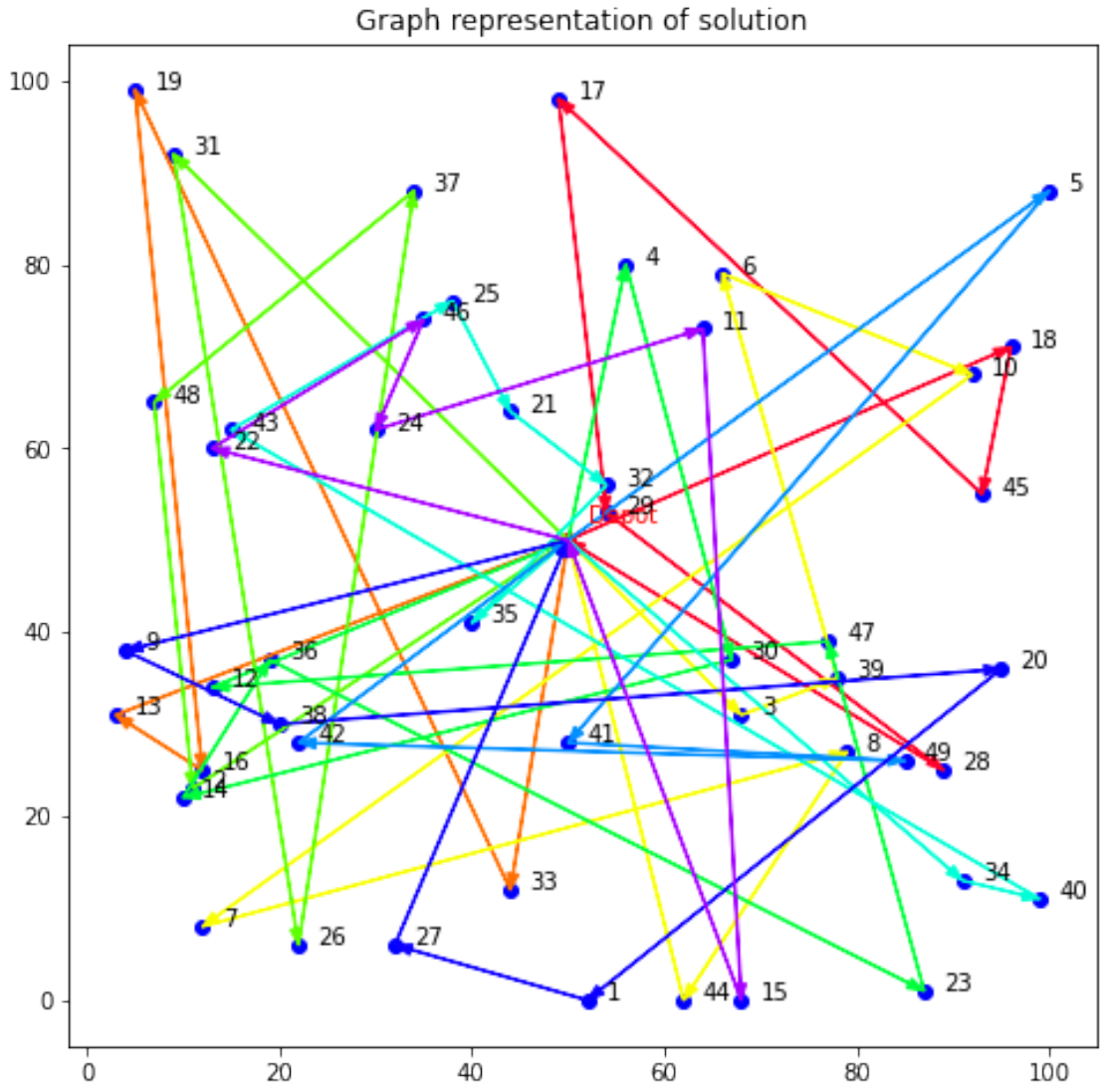
1.2 Tabu Search

[11]:
```
ts = TabuSearch(progress_bar=True, neighborhood_params=neighborhood_params)
tabu_sol = ts.fit_search(problem)
tabu_sol
```

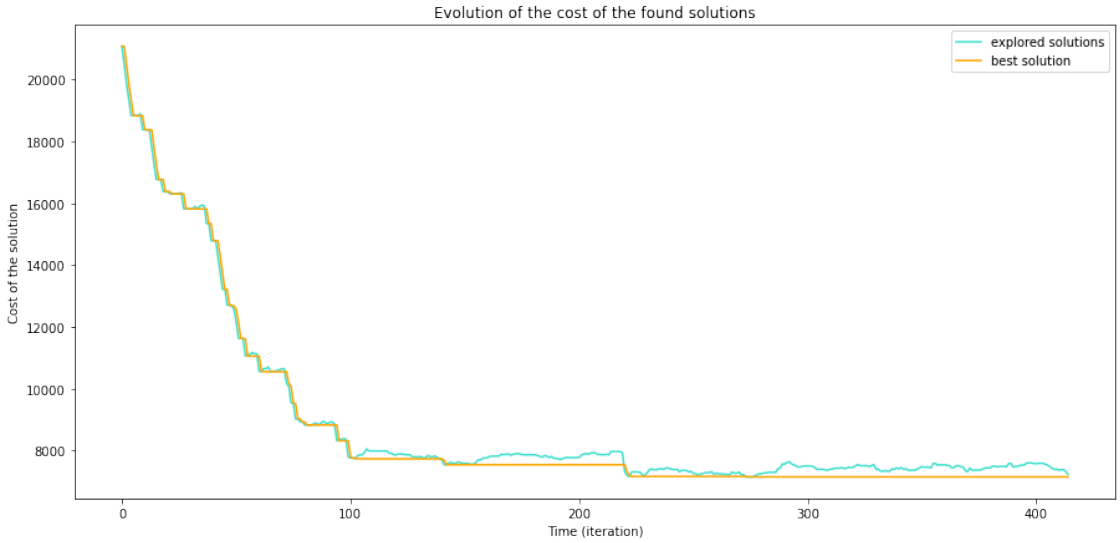
```
Cost: 7152.01: 100%|| 100/100 [00:00<00:00, 636.95it/s]
```

[11]:
```
[[0, 18, 45, 17, 29, 28, 0],
 [0, 33, 19, 16, 13, 0],
 [0, 3, 39, 6, 10, 7, 8, 44, 0],
 [0, 31, 26, 37, 48, 2, 0],
 [0, 4, 30, 14, 36, 23, 47, 12, 0],
 [0, 34, 40, 43, 25, 21, 32, 35, 0],
 [0, 5, 41, 49, 42, 0],
 [0, 9, 38, 20, 1, 27, 0],
 [0, 22, 46, 24, 11, 15, 0]]
```

[12]:
```
tabu_sol.plot_graph(figsize=8)
```



[13]:
```
ts.plot_evolution_cost(figsize=(15,7))
```



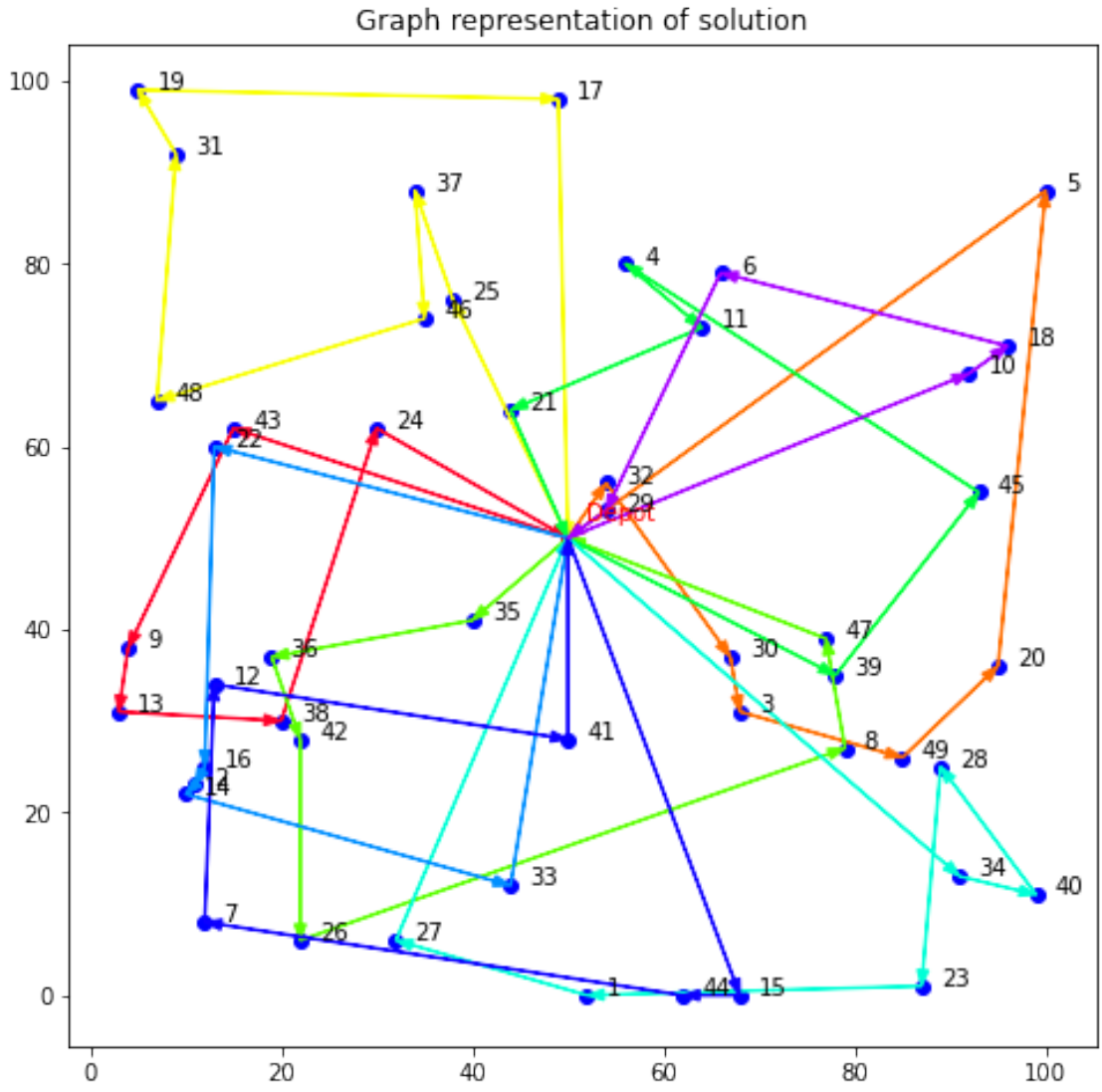
1.3 Genetic Algorithm

[14]:
```
ga = GeneticAlgorithm(num_evolu_per_search=100, rate_mutation=0.9,␣
↪progress_bar=True, neighborhood_params=neighborhood_params)
ga_sol = ga.fit_search(problem)
# print('cost =', ga_sol.cost())
ga_sol
```

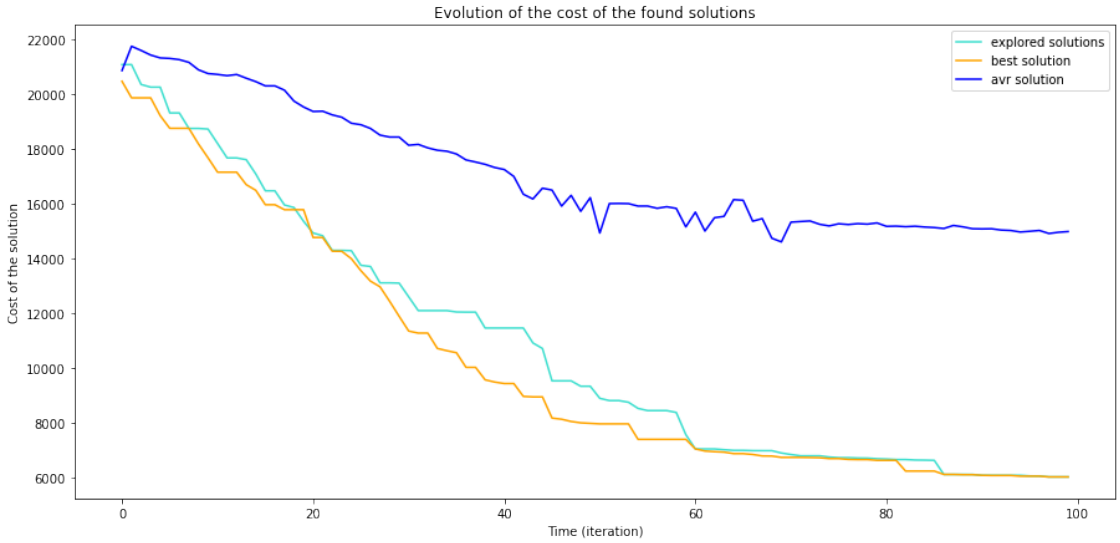
```
Cost: 6026.81: 100%|| 100/100 [02:23<00:00,  1.43s/it]
```

[14]:
```
[[0, 43, 9, 13, 38, 24, 0],
 [0, 32, 30, 3, 49, 20, 5, 0],
 [0, 25, 37, 46, 48, 31, 19, 17, 0],
 [0, 35, 36, 42, 26, 8, 47, 0],
 [0, 39, 45, 4, 11, 21, 0],
 [0, 34, 40, 28, 23, 1, 27, 0],
 [0, 22, 16, 2, 14, 33, 0],
 [0, 15, 44, 7, 12, 41, 0],
 [0, 10, 18, 6, 29, 0]]
```

[15]:
```
ga_sol.plot_graph(figsize=8)
```



[16]:
```
ga.plot_evolution_cost(figsize=(15,7))
```

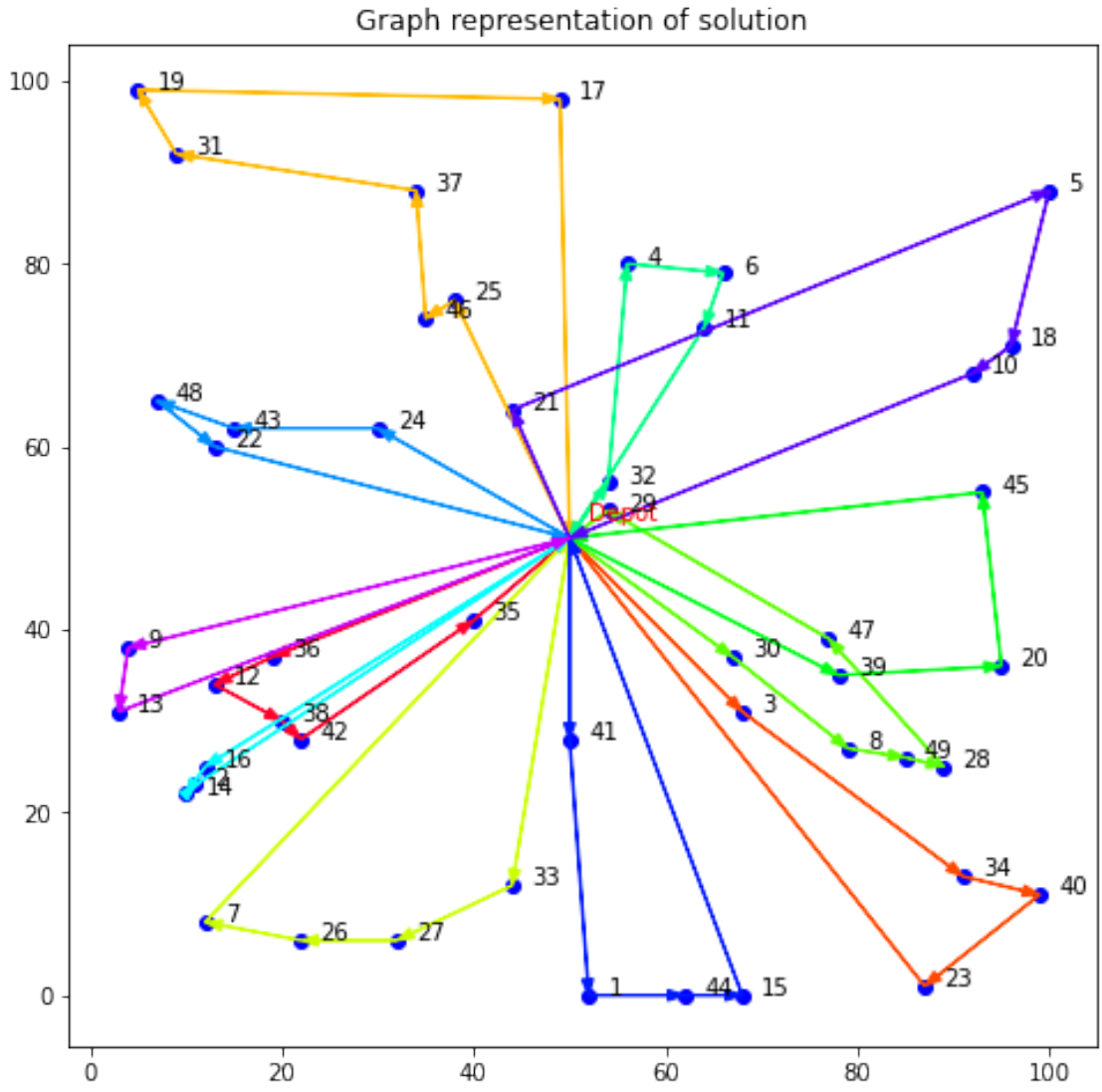


1.4 Variable Neighborhood Descent

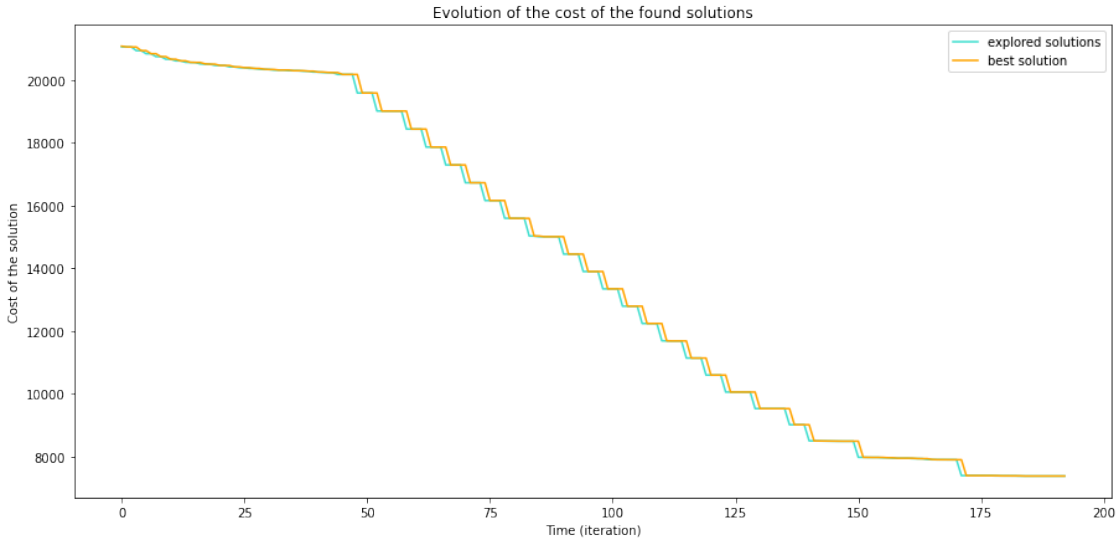
[17]:
```
vns = VariableNeighborhoodDescent(neighborhood_params=neighborhood_params)
vns_sol = vns.fit_search(problem)
print('cost of VNS solution found =', vns_sol.cost())
```

```
cost of VNS solution found = 7371.076954642759
```

[18]:
```
vns_sol.plot_graph(figsize=8)
```



[19]:
```
vns.plot_evolution_cost(figsize=(15,7))
```

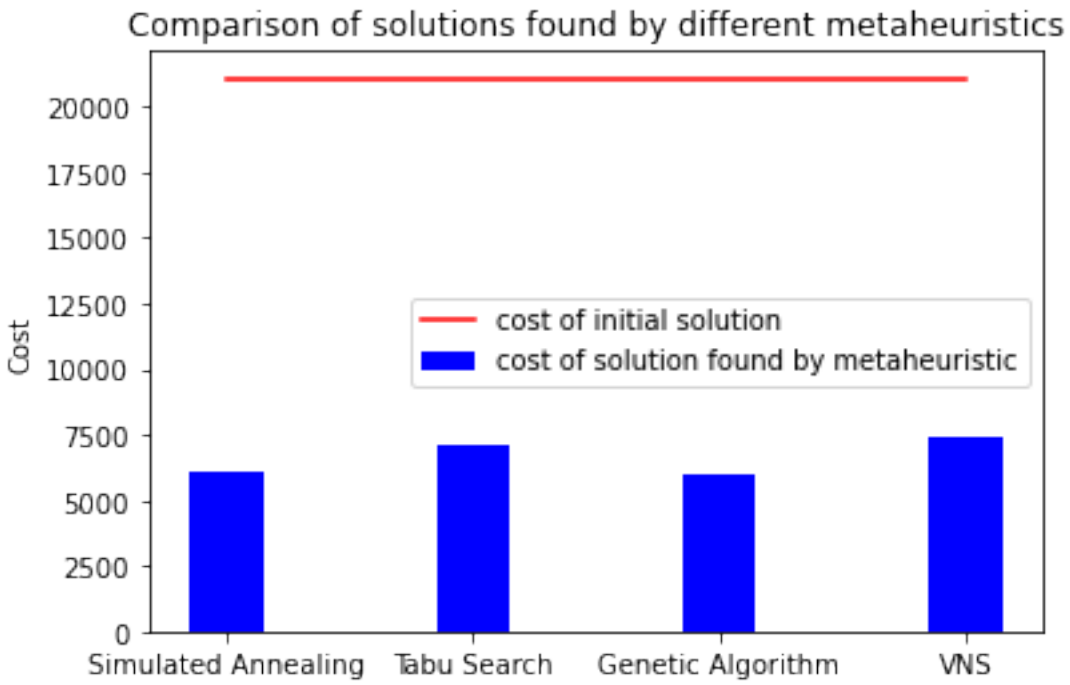


2 Comparaisons

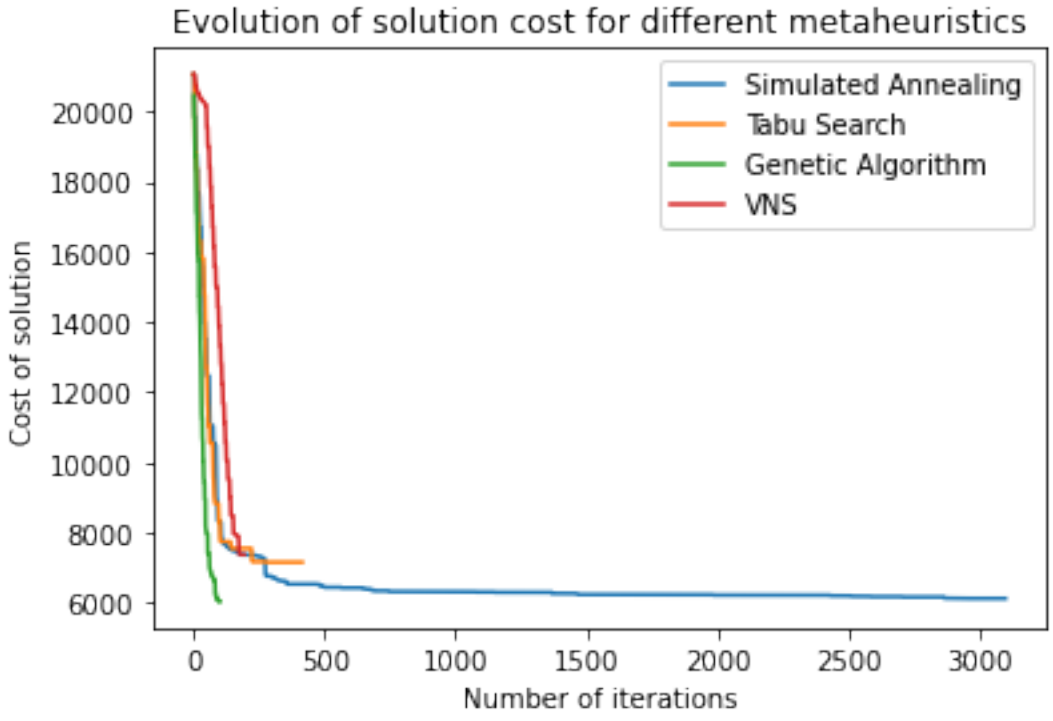
[20]:
```
def compare_solutions(init_sol, solutions, names):
    init_cost = init_sol.cost()
    costs = list(map(lambda s: s.cost(), solutions))
    plt.bar(names, costs, color='b', width=0.3, label='cost of solution found by␣
 ↪metaheuristic')
    plt.plot([init_cost]*len(solutions), c='r', label='cost of initial solution')
    plt.title('Comparison of solutions found by different metaheuristics')
    plt.ylabel('Cost')
    plt.legend()

def plot_models_evolution(models, names, crop_until=None):
    for model, name in zip(models, names):
        plt.plot(model.evolution_best_solution, label=name)
    plt.xlabel('Number of iterations')
    plt.ylabel('Cost of solution')
    plt.title('Evolution of solution cost for different metaheuristics')
    plt.legend()
```

[21]:
```
solutions = [rs_sol, tabu_sol, ga_sol, vns_sol]
models = [rs, ts, ga, vns]
names = ['Simulated Annealing', 'Tabu Search', 'Genetic Algorithm', 'VNS']
```

[22]:
```
compare_solutions(init_sol, solutions, names)
```



[23]:
```
plot_models_evolution(models, names)
```



[ ]: