

PEC 2

Pablo Riutort Grande

December 25, 2022

M0.536 - Optimización Metaheurística

MU Ingeniería Informática / MU Ingeniería Computacional y Matemática

Estudios de Informática, Multimedia y Telecomunicación

1 *ILS and PFSPs*

1.1 *Construct your own Python program to implement an ILS algorithm for solving the TSP.*

```
[1]: from math import sqrt
from random import randrange
from time import time

def timeit(func):
    def wrap_func(*args, **kwargs):
        t1 = time()
        result = func(*args, **kwargs)
        t2 = time()
        print(f'Function {func.__name__!r} with max iterations: {args[0]}, max_
↳improve: {args[1]} was executed in {(t2-t1):.4f}s\n')
        return result
    return wrap_func

def euclidean_distance(x,y):
    return sqrt(sum(pow((xi-yi),2) for xi, yi in zip(x,y)))

def tour_cost(permutation):
    total_distance = 0.0
    size = len(permutation)
    for index in range(size):
        start_node = permutation[index]
        end_node = permutation[0 if index == size-1 else index+1]
```

```

        total_distance += euclidean_distance(start_node, end_node)
    return total_distance

def stochastic_two_opt(permutation):
    """Delete two edges and reverse sequence in between them."""
    result = permutation[:]
    size = len(result)
    # select two random points
    p1, p2 = randrange(0, size), randrange(0, size)
    exclude = set([p1])
    if p1 == 0:
        exclude.add(size-1)
    else:
        exclude.add(p1-1)

    if p1 == size-1:
        exclude.add(0)
    else:
        exclude.add(p1+1)

    while p2 in exclude:
        p2 = randrange(0, size)

    if p2 < p1:
        p1, p2 = p2, p1

    result[p1:p2] = reversed(result[p1:p2])
    return result

def local_search(solution, cost, max_iterations,
    ↪search_function=stochastic_two_opt):
    for _ in reversed(range(max_iterations)):
        new_solution = search_function(solution)
        new_cost = tour_cost(new_solution)
        if new_cost < cost:
            solution = new_solution
            cost = new_cost
    return solution, cost

def perturbation(solution):
    new_solution = double_bridge_move(solution)
    new_cost = tour_cost(new_solution)
    return new_solution, new_cost

```

```

def double_bridge_move(permutation, slices=4):
    """Combine slices of permutation in order.

    The double-bridge move involves partitioning a permutation
    into 4 pieces (a,b,c,d) and putting it back together in a
    specific and jumbled ordering (a,d,c,b).
    """
    slice_length = len(permutation) / slices
    p1 = 1 + randrange(0, slice_length)
    p2 = p1 + 1 + randrange(0, slice_length)
    p3 = p2 + 1 + randrange(0, slice_length)
    return permutation[0:p1] + permutation[p3:] + permutation[p2:p3] +
    permutation[p1:p2]

def initial_solution(permutation):
    perm = permutation[:]
    size = len(perm)
    for index in range(size):
        shuffle_index = randrange(index, size)
        perm[shuffle_index] = perm[index]
        perm[index] = perm[shuffle_index]
    return perm

@timeit
def ils(max_iterations, max_improve, data):
    best_solution = initial_solution(data)
    best_cost = tour_cost(best_solution)

    best_solution, best_cost = local_search(best_solution, best_cost,
    max_improve)

    for iteration in reversed(range(max_iterations)):
        solution, cost = perturbation(best_solution)
        solution, cost = local_search(solution, cost, max_improve)
        if cost < best_cost:
            best_solution = solution
            best_cost = cost

    print(f"Best cost: {best_cost}. Best solution: {best_solution}")

```

```
[2]: from datasets import berlin_52 as input_tsp
```

```
max_iterations = [100, 1000]
```

```

max_improves = [10, 50]

for max_it in max_iterations:
    for max_im in max_improves:
        ils(max_it, max_im, input_tsp)

```

/tmp/ipykernel_7361/764429328.py:81: DeprecationWarning: non-integer arguments to randrange() have been deprecated since Python 3.10 and will be removed in a subsequent version

```
p1 = 1 + randrange(0, slice_length)
```

/tmp/ipykernel_7361/764429328.py:82: DeprecationWarning: non-integer arguments to randrange() have been deprecated since Python 3.10 and will be removed in a subsequent version

```
p2 = p1 + 1 + randrange(0, slice_length)
```

/tmp/ipykernel_7361/764429328.py:83: DeprecationWarning: non-integer arguments to randrange() have been deprecated since Python 3.10 and will be removed in a subsequent version

```
p3 = p2 + 1 + randrange(0, slice_length)
```

Best cost: 11960.818676659595. Best solution: [[1320, 315], [1320, 315], [1320, 315], [1220, 580], [565, 575], [420, 555], [25, 185], [25, 230], [25, 185], [415, 635], [145, 665], [145, 665], [145, 665], [555, 815], [25, 230], [560, 365], [555, 815], [845, 680], [945, 685], [555, 815], [525, 1000], [580, 1175], [880, 660], [845, 655], [835, 625], [650, 1130], [650, 1130], [700, 580], [770, 610], [795, 645], [880, 660], [685, 610], [345, 750], [345, 750], [560, 365], [560, 365], [660, 180], [795, 645], [565, 575], [565, 575], [685, 610], [795, 645], [795, 645], [830, 485], [835, 625], [845, 680], [880, 660], [835, 625], [975, 580], [1320, 315], [1530, 5], [1465, 200]]

Function 'ils' with max iterations: 100, max improve: 10 was executed in 0.1439s

Best cost: 9243.591490025025. Best solution: [[880, 660], [845, 680], [700, 580], [565, 575], [25, 185], [95, 260], [95, 260], [25, 230], [25, 230], [420, 555], [420, 555], [420, 555], [420, 555], [420, 555], [420, 555], [420, 555], [420, 555], [580, 1175], [525, 1000], [525, 1000], [525, 1000], [525, 1000], [525, 1000], [525, 1000], [650, 1130], [1150, 1160], [1150, 1160], [1605, 620], [1215, 245], [1530, 5], [1530, 5], [1465, 200], [1220, 580], [1220, 580], [1250, 400], [700, 580], [770, 610], [510, 875], [345, 750], [415, 635], [725, 370], [560, 365], [845, 655], [845, 655], [845, 655], [845, 655], [945, 685], [520, 585], [560, 365], [560, 365], [835, 625]]

Function 'ils' with max iterations: 100, max improve: 50 was executed in 0.5358s

Best cost: 7021.305660312526. Best solution: [[770, 610], [575, 665], [565, 575], [520, 585], [520, 585], [565, 575], [565, 575], [725, 370], [725, 370], [725, 370], [660, 180], [660, 180], [560, 365], [300, 465], [300, 465], [25, 230], [25, 230], [25, 230], [25, 230], [25, 185], [415, 635], [300, 465], [300, 465], [300, 465], [525, 1000], [650, 1130], [580, 1175], [525, 1000], [525, 1000], [510, 875], [510, 875], [770, 610], [770, 610], [700, 580], [770, 610], [770, 610], [845, 680], [1220, 580], [1530, 5], [1530, 5], [1530, 5], [1320,

315], [1215, 245], [975, 580], [845, 680], [845, 655], [880, 660], [945, 685], [975, 580], [880, 660], [720, 635], [770, 610]]

Function 'ils' with max iterations: 1000, max improve: 10 was executed in 1.0002s

Best cost: 7008.242065438563. Best solution: [[560, 365], [725, 370], [725, 370], [835, 625], [760, 650], [760, 650], [945, 685], [880, 660], [880, 660], [880, 660], [975, 580], [1605, 620], [1605, 620], [1605, 620], [1605, 620], [1605, 620], [1530, 5], [1530, 5], [1530, 5], [1530, 5], [1465, 200], [1215, 245], [1215, 245], [1215, 245], [945, 685], [945, 685], [845, 655], [685, 595], [685, 610], [770, 610], [565, 575], [415, 635], [420, 555], [300, 465], [25, 230], [25, 230], [25, 185], [25, 230], [145, 665], [580, 1175], [525, 1000], [510, 875], [415, 635], [565, 575], [685, 595], [685, 610], [685, 610], [560, 365], [560, 365], [560, 365], [480, 415], [480, 415]]

Function 'ils' with max iterations: 1000, max improve: 50 was executed in 4.5618s

1.2 Construct your own Python program to implement a TS algorithm for solving the TSP.

```
[3]: from math import sqrt
from random import randrange
from time import time

def timeit(func):
    def wrap_func(*args, **kwargs):
        t1 = time()
        result = func(*args, **kwargs)
        t2 = time()
        print(f'Function {func.__name__!r} with max new solutions: {args[0]},
↳max iterations: {args[1]} was executed in {(t2-t1):.4f}s\n')
        return result
    return wrap_func

def locate_best_new_sol(new_solutions):
    new_solutions.sort(key=lambda c: c["cost"])
    return new_solutions[0]

def is_tabu(permutation, tabu_list):
    size = len(permutation)
    for index, node in enumerate(permutation):
        next_node = [permutation[0] if index == size - 1 else
↳permutation[index+1]]
```

```

        edge = [node, next_node]
        if edge in tabu_list:
            return True
    return False

def euclidean_distance(start_node, end_node):
    _sum = 0.0
    for xi, yi in zip(start_node, end_node):
        _sum += pow((xi - yi), 2)
    return sqrt(_sum)

def stochastic_two_opt_with_edges(permutation):
    result = permutation[:]
    size = len(result)
    p1, p2 = randrange(0, size), randrange(0, size)
    exclude = set([p1])
    exclude.add(size - 1 if p1 == 0 else p1 - 1)
    exclude.add(0 if p1 == size - 1 else p1 + 1)
    while p2 in exclude:
        p2 = randrange(0, size)

    if p2 < p1:
        p1, p2 = p2, p1

    result[p1:p2] = reversed(result[p1:p2])
    return result, [[permutation[p1-1], permutation[p1]], [permutation[p2-1],
↪permutation[p2]]]

def generate_new_solution(base_solution, best_solution, tabu_list):
    new_permutation, edges, new_solution = None, None, {}
    while new_permutation == None or is_tabu(new_permutation, tabu_list):
        new_permutation, edges = ↪
↪stochastic_two_opt_with_edges(base_solution["permutation"])
        if tour_cost(new_permutation) < best_solution["cost"]:
            break

    new_solution["permutation"] = new_permutation
    new_solution["cost"] = tour_cost(new_solution["permutation"])
    new_solution["edges"] = edges
    return new_solution

def construct_initial_solution(init_permutation):

```

```

    permutation = init_permutation[:]
    size = len(permutation)
    for index in range(size):
        shuffle_index = randrange(index, size)
        permutation[shuffle_index], permutation[index] = permutation[index],
↪ permutation[shuffle_index]
    return permutation

def tour_cost(permutation):
    total_distance = 0.0
    size = len(permutation)
    for index in range(size):
        start_node = permutation[index]
        end_node = permutation[0] if index == size - 1 else permutation[index +
↪ 1]
        total_distance += euclidean_distance(start_node, end_node)
    return total_distance

@timeit
def tbs(max_new_solutions, max_iterations, data, max_edges_in_tabu_list=10,
↪ k=5):
    best_solution = {
        "edges": None,
        "permutation": construct_initial_solution(data)
    }
    best_solution["cost"] = tour_cost(best_solution["permutation"])
    base_solution = best_solution

    credit = 0
    tabu_list = []

    while max_iterations > 0:
        new_solutions = []
        for index in range(0, max_new_solutions):
            new_solution = generate_new_solution(base_solution, best_solution,
↪ tabu_list)
            new_solutions.append(new_solution)

        best_new_solution = locate_best_new_sol(new_solutions)
        delta = best_new_solution["cost"] - base_solution["cost"]
        if delta <= 0:
            credit = -1 * delta
            base_solution = best_new_solution

            if best_new_solution["cost"] < best_solution["cost"]:

```

```

        best_solution = best_new_solution
        print(f"it: {max_iterations} cost: {best_solution['cost']:.2f}")
        for edge in best_new_solution["edges"]:
            tabu_list.append(edge)
            if len(tabu_list) > max_edges_in_tabu_list:
                del tabu_list[0]
    else:
        if delta <= k * credit:
            credit = 0
            base_solution = best_new_solution
        max_iterations -= 1

```

```

[4]: from datasets import berlin_52 as input_tsp

max_new_solutions = [20, 60]
max_iterations = [1000, 5000]
for max_new_solution in max_new_solutions:
    for max_iteration in max_iterations:
        tbs(max_new_solution, max_iteration, input_tsp)

```

```

it: 1000 cost: 29031.76
it: 999 cost: 28166.48
it: 998 cost: 26988.65
it: 997 cost: 26327.61
it: 996 cost: 26197.18
it: 995 cost: 25786.23
it: 994 cost: 25356.14
it: 993 cost: 24979.88
it: 992 cost: 24309.78
it: 991 cost: 23269.08
it: 990 cost: 22982.12
it: 989 cost: 22652.56
it: 988 cost: 22232.90
it: 987 cost: 21348.61
it: 986 cost: 20767.94
it: 985 cost: 20402.49
it: 984 cost: 20139.47
it: 983 cost: 19792.61
it: 982 cost: 19385.02
it: 981 cost: 18986.45
it: 980 cost: 18457.59
it: 979 cost: 18356.52
it: 978 cost: 18222.94
it: 977 cost: 17918.23
it: 976 cost: 17234.99
it: 975 cost: 17005.30

```


it: 974 cost: 16495.42
it: 973 cost: 16475.30
it: 972 cost: 16396.17
it: 971 cost: 16088.49
it: 970 cost: 15955.82
it: 969 cost: 15384.74
it: 968 cost: 15377.30
it: 967 cost: 15216.32
it: 966 cost: 14747.91
it: 964 cost: 14625.94
it: 963 cost: 14572.38
it: 962 cost: 14488.24
it: 961 cost: 14167.90
it: 960 cost: 13694.68
it: 959 cost: 13641.05
it: 958 cost: 13589.20
it: 957 cost: 13290.61
it: 956 cost: 13237.61
it: 955 cost: 13046.80
it: 954 cost: 13036.01
it: 953 cost: 12885.59
it: 952 cost: 12756.09
it: 950 cost: 12733.75
it: 948 cost: 12485.84
it: 943 cost: 12201.04
it: 942 cost: 12186.65
it: 941 cost: 12171.99
it: 940 cost: 12122.22
it: 937 cost: 12050.02
it: 934 cost: 11545.36
it: 933 cost: 11534.14
it: 931 cost: 11327.96
it: 923 cost: 11319.90
it: 919 cost: 11294.37
it: 918 cost: 11010.31
it: 917 cost: 10877.67
it: 916 cost: 10860.24
it: 914 cost: 10817.91
it: 909 cost: 10798.72
it: 905 cost: 10696.43
it: 904 cost: 10687.69
it: 902 cost: 10635.19
it: 901 cost: 10571.42
it: 900 cost: 10452.97
it: 899 cost: 10373.90
it: 897 cost: 10144.38
it: 896 cost: 9972.72
it: 889 cost: 9891.85

it: 888 cost: 9871.81
it: 886 cost: 9780.66
it: 881 cost: 9774.89
it: 871 cost: 9771.78
it: 867 cost: 9678.79
it: 855 cost: 9543.67
it: 852 cost: 9539.73
it: 851 cost: 9444.18
it: 850 cost: 9427.82
it: 848 cost: 9425.97
it: 847 cost: 9233.60
it: 798 cost: 9188.53
it: 797 cost: 9137.22
it: 796 cost: 9101.33
it: 788 cost: 9058.39
it: 712 cost: 9044.80
it: 665 cost: 8899.82
it: 661 cost: 8834.21
it: 660 cost: 8731.67
it: 656 cost: 8675.74
it: 642 cost: 8603.70
it: 636 cost: 8581.75
it: 618 cost: 8576.11
it: 607 cost: 8527.86
it: 551 cost: 8508.47
it: 539 cost: 8449.78
it: 537 cost: 8389.35
it: 500 cost: 8368.14
it: 342 cost: 8292.49
it: 315 cost: 8276.60
it: 314 cost: 8275.14
it: 306 cost: 8256.02
it: 285 cost: 8184.40
it: 142 cost: 8138.81
it: 98 cost: 8100.76
it: 95 cost: 8091.82

Function 'tbs' with max new solutions: 20, max iterations: 1000 was executed in 3.7422s

it: 5000 cost: 27776.09
it: 4999 cost: 26958.56
it: 4998 cost: 26564.10
it: 4997 cost: 25816.66
it: 4996 cost: 25089.92
it: 4995 cost: 24655.61
it: 4994 cost: 24075.43
it: 4993 cost: 23706.24
it: 4992 cost: 23065.96

it: 4991 cost: 22873.04
it: 4990 cost: 22318.98
it: 4989 cost: 21857.63
it: 4988 cost: 21604.97
it: 4987 cost: 21340.17
it: 4986 cost: 21090.43
it: 4985 cost: 20737.97
it: 4984 cost: 20340.62
it: 4983 cost: 19935.39
it: 4982 cost: 19647.47
it: 4981 cost: 19366.84
it: 4980 cost: 18939.23
it: 4979 cost: 18684.54
it: 4978 cost: 18483.10
it: 4977 cost: 18245.48
it: 4976 cost: 18158.57
it: 4975 cost: 17564.98
it: 4974 cost: 16452.56
it: 4973 cost: 15857.06
it: 4972 cost: 15780.71
it: 4971 cost: 15588.82
it: 4970 cost: 15447.58
it: 4969 cost: 15128.24
it: 4968 cost: 14670.61
it: 4967 cost: 14667.28
it: 4966 cost: 14286.73
it: 4965 cost: 14083.00
it: 4964 cost: 13582.62
it: 4963 cost: 13345.88
it: 4962 cost: 13185.46
it: 4961 cost: 13151.66
it: 4960 cost: 13108.49
it: 4958 cost: 13049.75
it: 4955 cost: 12882.97
it: 4954 cost: 12867.89
it: 4953 cost: 12828.94
it: 4951 cost: 12809.92
it: 4950 cost: 12454.89
it: 4949 cost: 12448.12
it: 4948 cost: 12336.14
it: 4947 cost: 12191.63
it: 4946 cost: 12054.17
it: 4945 cost: 12051.16
it: 4943 cost: 11976.19
it: 4942 cost: 11954.95
it: 4940 cost: 11707.96
it: 4939 cost: 11571.66
it: 4938 cost: 11408.29

it: 4937 cost: 11275.21
it: 4936 cost: 11235.37
it: 4935 cost: 11118.64
it: 4933 cost: 11111.13
it: 4931 cost: 11081.89
it: 4929 cost: 10962.04
it: 4928 cost: 10853.39
it: 4925 cost: 10535.65
it: 4924 cost: 10526.53
it: 4922 cost: 10446.80
it: 4920 cost: 10414.69
it: 4911 cost: 10402.98
it: 4909 cost: 10322.74
it: 4906 cost: 10315.75
it: 4905 cost: 10271.54
it: 4903 cost: 10170.86
it: 4900 cost: 10167.30
it: 4899 cost: 10117.77
it: 4883 cost: 10098.88
it: 4882 cost: 9991.84
it: 4881 cost: 9957.92
it: 4876 cost: 9930.27
it: 4865 cost: 9816.96
it: 4864 cost: 9749.13
it: 4855 cost: 9709.52
it: 4853 cost: 9491.60
it: 4852 cost: 9326.58
it: 4807 cost: 9304.68
it: 4782 cost: 9256.03
it: 4774 cost: 9245.20
it: 4753 cost: 9035.25
it: 4744 cost: 8996.54
it: 4743 cost: 8970.90
it: 4740 cost: 8938.00
it: 4739 cost: 8888.27
it: 4738 cost: 8886.71
it: 4736 cost: 8879.24
it: 4735 cost: 8815.90
it: 4679 cost: 8753.34
it: 4664 cost: 8736.60
it: 4661 cost: 8602.52
it: 4642 cost: 8558.84
it: 4480 cost: 8484.80
it: 4436 cost: 8423.77
it: 4435 cost: 8316.82
it: 4418 cost: 8235.96
it: 3961 cost: 8050.59
it: 3896 cost: 7919.96

it: 3805 cost: 7917.85
it: 3792 cost: 7818.49
it: 3744 cost: 7731.81
it: 3731 cost: 7696.48
Function 'tbs' with max new solutions: 20, max iterations: 5000 was executed in
17.7116s

it: 1000 cost: 28666.20
it: 999 cost: 26847.77
it: 998 cost: 25688.59
it: 997 cost: 24548.61
it: 996 cost: 23807.08
it: 995 cost: 22784.63
it: 994 cost: 22074.42
it: 993 cost: 21499.04
it: 992 cost: 20470.18
it: 991 cost: 19421.13
it: 990 cost: 19086.45
it: 989 cost: 18449.09
it: 988 cost: 18058.21
it: 987 cost: 17479.54
it: 986 cost: 17125.38
it: 985 cost: 16820.30
it: 984 cost: 16261.29
it: 983 cost: 15979.55
it: 982 cost: 15775.45
it: 981 cost: 15027.73
it: 980 cost: 14858.73
it: 979 cost: 14789.40
it: 978 cost: 14377.43
it: 977 cost: 13877.55
it: 976 cost: 13648.39
it: 975 cost: 13296.50
it: 974 cost: 13214.48
it: 973 cost: 13033.06
it: 972 cost: 12914.14
it: 971 cost: 12319.47
it: 970 cost: 12011.59
it: 969 cost: 11710.62
it: 968 cost: 11425.73
it: 967 cost: 11316.88
it: 966 cost: 11221.12
it: 965 cost: 11150.48
it: 964 cost: 11093.31
it: 963 cost: 10918.52
it: 962 cost: 10488.27
it: 961 cost: 10437.02
it: 960 cost: 10284.55

it: 959 cost: 10186.65
it: 957 cost: 9879.59
it: 956 cost: 9804.49
it: 955 cost: 9775.33
it: 953 cost: 9565.82
it: 952 cost: 9408.63
it: 939 cost: 9235.59
it: 937 cost: 9200.69
it: 935 cost: 9123.96
it: 933 cost: 9069.52
it: 924 cost: 9065.14
it: 923 cost: 9039.69
it: 922 cost: 8916.31
it: 915 cost: 8869.19
it: 911 cost: 8845.44
it: 824 cost: 8844.57
it: 822 cost: 8841.37
it: 746 cost: 8810.67
it: 744 cost: 8789.16
it: 733 cost: 8780.95
it: 719 cost: 8769.91
it: 451 cost: 8664.79
it: 418 cost: 8664.54
it: 417 cost: 8643.99
it: 393 cost: 8625.57
it: 376 cost: 8605.79
it: 314 cost: 8586.95
it: 189 cost: 8555.21
it: 188 cost: 8530.23
it: 187 cost: 8489.25
it: 186 cost: 8440.09
it: 185 cost: 8429.05
it: 131 cost: 8416.80
it: 116 cost: 8392.90

Function 'tbs' with max new solutions: 60, max iterations: 1000 was executed in 10.7573s

it: 5000 cost: 29370.90
it: 4999 cost: 28340.61
it: 4998 cost: 27428.86
it: 4997 cost: 26806.85
it: 4996 cost: 25883.70
it: 4995 cost: 25506.28
it: 4994 cost: 23853.66
it: 4993 cost: 22958.46
it: 4992 cost: 22349.62
it: 4991 cost: 21860.01
it: 4990 cost: 20865.50

it: 4989 cost: 20346.17
it: 4988 cost: 19500.64
it: 4987 cost: 19035.42
it: 4986 cost: 18690.67
it: 4985 cost: 18305.26
it: 4984 cost: 17824.06
it: 4983 cost: 17605.29
it: 4982 cost: 17308.71
it: 4981 cost: 16700.30
it: 4980 cost: 16210.59
it: 4979 cost: 15854.47
it: 4978 cost: 15346.24
it: 4977 cost: 14811.26
it: 4976 cost: 14692.56
it: 4975 cost: 14509.64
it: 4974 cost: 14112.12
it: 4973 cost: 13865.17
it: 4972 cost: 13711.35
it: 4971 cost: 13577.01
it: 4970 cost: 13075.14
it: 4969 cost: 12805.72
it: 4968 cost: 12571.59
it: 4967 cost: 12486.46
it: 4966 cost: 12009.88
it: 4965 cost: 11794.82
it: 4964 cost: 11527.82
it: 4963 cost: 11507.86
it: 4962 cost: 11370.15
it: 4961 cost: 11244.21
it: 4960 cost: 11187.91
it: 4959 cost: 11184.82
it: 4958 cost: 11166.06
it: 4957 cost: 11042.09
it: 4956 cost: 10858.15
it: 4955 cost: 10831.23
it: 4954 cost: 10733.98
it: 4953 cost: 10553.67
it: 4952 cost: 10477.94
it: 4951 cost: 10408.37
it: 4950 cost: 10166.86
it: 4947 cost: 10115.70
it: 4946 cost: 9522.12
it: 4944 cost: 9476.25
it: 4943 cost: 9468.07
it: 4940 cost: 9427.73
it: 4938 cost: 9389.61
it: 4935 cost: 9252.78
it: 4932 cost: 9223.22

```
it: 4931 cost: 9221.60
it: 4930 cost: 9214.11
it: 4929 cost: 9173.53
it: 4928 cost: 9100.03
it: 4927 cost: 9047.02
it: 4926 cost: 8944.83
it: 4925 cost: 8933.97
it: 4923 cost: 8912.00
it: 4921 cost: 8878.08
it: 4913 cost: 8869.55
it: 4911 cost: 8847.85
it: 4906 cost: 8703.62
it: 4899 cost: 8694.94
it: 4897 cost: 8680.80
it: 4896 cost: 8639.82
it: 4895 cost: 8592.80
it: 4859 cost: 8517.30
it: 4784 cost: 8516.00
it: 4700 cost: 8494.99
it: 4699 cost: 8453.19
it: 4689 cost: 8449.94
it: 4673 cost: 8444.78
it: 4600 cost: 8402.93
it: 4598 cost: 8370.85
it: 4595 cost: 8353.76
it: 4594 cost: 8313.29
it: 4588 cost: 8303.66
it: 4575 cost: 8268.34
it: 4511 cost: 8267.34
it: 4479 cost: 8244.66
it: 4402 cost: 8230.66
it: 4345 cost: 8224.42
it: 4227 cost: 8217.34
```

Function 'tbs' with max new solutions: 60, max iterations: 5000 was executed in 61.2868s

1.3 *Construct your own Python program to implement the NEH heuristic for solving the PFSP and test it in different instances.*

```
[5]: class Job:
      def __init__(self, ID, processing_times, TPT):
          self.ID = ID
          self.processing_times = processing_times
          self.TPT = TPT
```



```

class Solution:
    last_ID = -1

    def __init__(self, n_jobs, n_machines):
        Solution.last_ID += 1
        self.ID = Solution.last_ID
        self.n_jobs = n_jobs
        self.n_machines = n_machines
        self.jobs = []
        self.makespan = 0.0

    def calculate_makespan(self):
        rows = self.n_jobs
        cols = self.n_machines
        times = [[0 for _ in range(cols)] for _ in range(rows)]
        for column in range(cols):
            for row in range(rows):
                if column == row == 0:
                    times[0][0] = self.jobs[0].processing_times[0]
                elif column == 0:
                    times[row][0] = times[row - 1][0] + self.jobs[row].
↪processing_times[0]
                elif row == 0:
                    times[0][column] = times[0][column - 1] + self.jobs[0].
↪processing_times[column]
                else:
                    max_time = max(times[row - 1][column], times[row][column - 1])
↪1))
                    times[row][column] = max_time + self.jobs[row].
↪processing_times[column]
            return times[rows - 1][cols - 1]

```

```

[6]: import operator

from time import time

from PSFP_elements import Job, Solution

def timeit(func):
    def wrap_func(*args, **kwargs):
        print(f"{args[1]} jobs and {args[2]} machines")
        t1 = time()
        result = func(*args, **kwargs)
        t2 = time()

```

```

        print(f'Function {func.__name__!r} computation time: {(t2-t1):.4f}s')
        return result
    return wrap_func

def read_instance(instance_name):
    file_name = f"data/{instance_name}_inputs.txt"
    jobs, n_jobs, n_machines = [], 0, 0
    with open(file_name) as instance:
        i = -3
        for line in instance:
            if i == -3: pass
            elif i == -2:
                n_jobs = int(line.split()[0])
                n_machines = int(line.split()[1])
            elif i == -1: pass
            else:
                data = [float(x) for x in line.split("\t")]
                jobs.append(Job(i, data, sum(data)))
                i += 1
    return jobs, n_jobs, n_machines

def calc_QMatrix(solution, k):
    rows = k + 1
    cols = n_machines
    q = [[0 for _ in range(cols)] for _ in range(rows)]
    for i in range(k, -1, -1):
        for j in range(n_machines - 1, -1, -1):
            if i == k:
                q[k][j] = 0
            elif i == k-1 and j == n_machines - 1:
                q[k-1][n_machines-1] = solution.jobs[k-1].
                ↪ processing_times[n_machines-1]
            elif j == n_machines - 1:
                q[i][n_machines - 1] = q[i+1][n_machines-1] + solution.jobs[i].
                ↪ processing_times[n_machines-1]
            elif i == k - 1:
                q[k-1][j] = q[k-1][j+1] + solution.jobs[k-1].processing_times[j]
            else:
                max_time = max(q[i+1][j], q[i][j+1])
                q[i][j] = max_time + solution.jobs[i].processing_times[j]
    return q

def calc_FMatrix(solution, k, e):
    rows = k + 1

```

```

cols = n_machines
f = [[0 for _ in range(cols)] for _ in range(rows)]
for i in range(k+1):
    for j in range(n_machines):
        if i == j == 0:
            f[0][0] = solution.jobs[k].processing_times[0]
        elif j == 0:
            f[i][0] = e[i-1][0] + solution.jobs[k].processing_times[0]
        elif i == 0:
            f[0][j] = f[0][j-1] + solution.jobs[k].processing_times[j]
        else:
            max_time = max(e[i-1][j], f[i][j-1])
            f[i][j] = max_time + solution.jobs[k].processing_times[j]
    return f

def calc_EMatrix(solution, k):
    rows = k
    cols = n_machines
    e = [[0 for _ in range(cols)] for _ in range(rows)]
    for i in range(k):
        for j in range(n_machines):
            if i == j == 0:
                e[0][0] = solution.jobs[0].processing_times[0]
            elif j == 0:
                e[i][0] = e[i-1][0] + solution.jobs[i].processing_times[0]
            elif i == 0:
                e[0][j] = e[0][j-1] + solution.jobs[0].processing_times[j]
            else:
                max_time = max(e[i-1][j], e[i][j-1])
                e[i][j] = max_time + solution.jobs[i].processing_times[j]
    return e

def improve_by_shifting_job_to_left(solution, k):
    best_position = k
    min_makespan = float("inf")
    e_matrix = calc_EMatrix(solution, k)
    q_matrix = calc_QMatrix(solution, k)
    f_matrix = calc_FMatrix(solution, k, e_matrix)
    for i in range(k, -1, -1):
        max_sum = 0.0
        for j in range(solution.n_machines):
            new_sum = f_matrix[i][j] + q_matrix[i][j]
            if new_sum > max_sum: max_sum = new_sum
        new_makespan = max_sum

```

```

        if new_makespan <= min_makespan:
            min_makespan = new_makespan
            best_position = i

    if best_position < k:
        job = solution.jobs[k]
        for i in range(k, best_position, -1):
            solution.jobs[i] = solution.jobs[i-1]
        solution.jobs[best_position] = job
    if k == solution.n_jobs - 1:
        solution.makespan = min_makespan
    return solution

@timeit
def neh(jobs, n_jobs, n_machines):
    jobs.sort(key=operator.attrgetter("TPT"), reverse=True)
    solution = Solution(n_jobs, n_machines)
    for index in range(n_jobs):
        solution.jobs.append(jobs[index])
        solution = improve_by_shifting_job_to_left(solution, index)
    print(f"NEH makespan with Taillard acceleration {solution.makespan:.2f}")
    print(f"NEH verification with traditional method: {solution.
↵calculate_makespan():.2f}")

```

```

[7]: instances = ["tai117_500_20", "tai079_100_10", "tai044_50_10", "tai002_20_5"]
for instance in instances:
    print(f"Instance {instance}")
    jobs, n_jobs, n_machines = read_instance(instance)
    neh(jobs, n_jobs, n_machines)
    print(f"Solution: {[str(job.ID) for job in jobs]}\n")

```

```

Instance tai117_500_20
500 jobs and 20 machines
NEH makespan with Taillard acceleration 26797.00
NEH verification with traditional method: 26797.00
Function 'neh' computation time: 5.9367s
Solution: ['130', '40', '58', '388', '270', '198', '467', '274', '261', '293',
'203', '332', '438', '185', '146', '27', '210', '328', '66', '219', '100', '18',
'71', '391', '249', '142', '497', '373', '158', '340', '441', '408', '313',
'448', '245', '399', '257', '65', '123', '128', '147', '163', '459', '347',
'434', '11', '57', '310', '484', '460', '107', '206', '254', '166', '353',
'124', '412', '427', '31', '86', '99', '182', '73', '473', '285', '129', '413',
'366', '0', '157', '64', '360', '9', '423', '234', '462', '341', '271', '28',
'119', '54', '74', '220', '479', '207', '214', '239', '382', '456', '273',
'287', '165', '326', '401', '89', '138', '280', '398', '133', '45', '290',
'267', '204', '369', '403', '180', '410', '8', '98', '283', '301', '371', '108',

```

'487', '452', '49', '308', '361', '393', '428', '451', '392', '14', '120',
 '217', '454', '486', '488', '29', '76', '85', '82', '279', '23', '39', '192',
 '442', '477', '63', '358', '367', '303', '125', '174', '244', '278', '136',
 '465', '466', '34', '179', '197', '316', '407', '491', '13', '52', '296', '121',
 '62', '201', '226', '431', '317', '145', '20', '417', '436', '187', '268',
 '377', '75', '159', '60', '337', '148', '183', '299', '314', '449', '41', '46',
 '470', '43', '255', '295', '495', '42', '90', '496', '117', '169', '91', '188',
 '22', '394', '242', '429', '53', '315', '307', '333', '149', '105', '26', '110',
 '103', '411', '269', '79', '196', '208', '247', '190', '432', '238', '372',
 '339', '416', '424', '253', '223', '327', '489', '370', '178', '143', '152',
 '378', '35', '205', '335', '171', '232', '19', '199', '72', '80', '228', '439',
 '153', '345', '24', '32', '302', '445', '50', '425', '474', '330', '154', '292',
 '440', '444', '450', '132', '231', '252', '70', '38', '81', '83', '131', '37',
 '379', '334', '30', '47', '78', '365', '141', '200', '375', '475', '263', '346',
 '357', '33', '195', '464', '191', '469', '248', '390', '93', '59', '113', '322',
 '127', '155', '498', '36', '281', '338', '402', '396', '420', '55', '385',
 '387', '405', '363', '419', '356', '381', '126', '458', '175', '265', '348',
 '213', '284', '364', '490', '221', '170', '224', '151', '168', '306', '350',
 '48', '3', '67', '355', '493', '482', '297', '480', '481', '134', '368', '1',
 '102', '150', '162', '176', '397', '44', '305', '342', '94', '447', '225',
 '320', '266', '483', '235', '156', '275', '260', '262', '289', '164', '161',
 '181', '457', '56', '422', '215', '453', '116', '135', '216', '237', '172',
 '16', '194', '122', '211', '291', '96', '324', '472', '352', '492', '25', '51',
 '276', '336', '272', '240', '97', '298', '140', '177', '325', '184', '84',
 '106', '359', '101', '282', '88', '173', '6', '437', '139', '227', '418', '494',
 '374', '68', '167', '351', '400', '104', '414', '461', '109', '329', '349',
 '230', '77', '189', '229', '222', '10', '137', '202', '286', '12', '2', '404',
 '343', '344', '406', '294', '435', '160', '21', '15', '92', '236', '304', '380',
 '114', '193', '395', '144', '264', '258', '476', '409', '288', '115', '318',
 '5', '386', '69', '312', '309', '443', '499', '209', '256', '415', '433', '95',
 '233', '218', '186', '7', '112', '111', '362', '376', '251', '118', '241',
 '250', '87', '383', '421', '485', '430', '246', '463', '300', '212', '17',
 '426', '446', '311', '4', '478', '471', '319', '389', '331', '259', '323', '61',
 '455', '277', '468', '243', '354', '321', '384']

Instance tai079_100_10

100 jobs and 10 machines

NEH makespan with Taillard acceleration 6032.00

NEH verification with traditional method: 6032.00

Function 'neh' computation time: 0.1090s

Solution: ['12', '21', '13', '27', '62', '34', '77', '92', '70', '49', '81',
 '38', '88', '68', '15', '7', '31', '40', '29', '6', '54', '11', '46', '87',
 '71', '57', '44', '58', '26', '99', '30', '55', '60', '69', '19', '39', '50',
 '76', '14', '80', '24', '25', '78', '75', '43', '9', '4', '32', '56', '85',
 '36', '89', '93', '51', '10', '96', '16', '23', '79', '28', '41', '65', '67',
 '84', '82', '0', '95', '48', '83', '3', '73', '47', '64', '1', '61', '33', '66',
 '5', '59', '22', '98', '2', '52', '74', '17', '35', '37', '86', '97', '20',
 '18', '72', '8', '53', '94', '42', '91', '45', '63', '90']

```

Instance tai044_50_10
50 jobs and 10 machines
NEH makespan with Taillard acceleration 3198.00
NEH verification with traditional method: 3198.00
Function 'neh' computation time: 0.0281s
Solution: ['46', '14', '42', '11', '25', '13', '3', '34', '1', '31', '37', '10',
'0', '40', '20', '5', '22', '48', '29', '24', '35', '15', '39', '49', '47',
'23', '27', '21', '16', '38', '33', '30', '2', '7', '44', '26', '36', '17', '6',
'12', '45', '32', '28', '18', '9', '4', '43', '41', '8', '19']

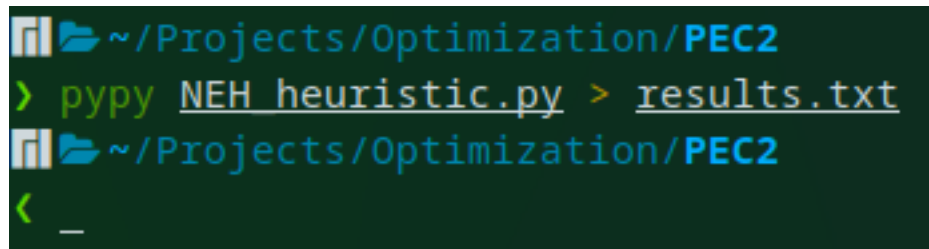
```

```

Instance tai002_20_5
20 jobs and 5 machines
NEH makespan with Taillard acceleration 1365.00
NEH verification with traditional method: 1365.00
Function 'neh' computation time: 0.0025s
Solution: ['12', '0', '4', '10', '6', '8', '1', '11', '7', '3', '19', '15',
'14', '5', '16', '2', '17', '18', '9', '13']

```

1.4 Use PyPy to speed up your code.



```

~/Projects/Optimization/PEC2
> pypy NEH heuristic.py > results.txt
~/Projects/Optimization/PEC2
< _

```

```
[8]: ! cat results.txt
```

```

Instance tai117_500_20
500 jobs and 20 machines
NEH makespan with Taillard acceleration 26797.00
NEH verification with traditional method: 26797.00
Function 'neh' computation time: 0.5086s
Solution: ['130', '40', '58', '388', '270', '198', '467', '274', '261', '293',
'203', '332', '438', '185', '146', '27', '210', '328', '66', '219', '100', '18',
'71', '391', '249', '142', '497', '373', '158', '340', '441', '408', '313',
'448', '245', '399', '257', '65', '123', '128', '147', '163', '459', '347',
'434', '11', '57', '310', '484', '460', '107', '206', '254', '166', '353',
'124', '412', '427', '31', '86', '99', '182', '73', '473', '285', '129', '413',
'366', '0', '157', '64', '360', '9', '423', '234', '462', '341', '271', '28',
'119', '54', '74', '220', '479', '207', '214', '239', '382', '456', '273',
'287', '165', '326', '401', '89', '138', '280', '398', '133', '45', '290',
'267', '204', '369', '403', '180', '410', '8', '98', '283', '301', '371', '108',

```

'487', '452', '49', '308', '361', '393', '428', '451', '392', '14', '120',
 '217', '454', '486', '488', '29', '76', '85', '82', '279', '23', '39', '192',
 '442', '477', '63', '358', '367', '303', '125', '174', '244', '278', '136',
 '465', '466', '34', '179', '197', '316', '407', '491', '13', '52', '296', '121',
 '62', '201', '226', '431', '317', '145', '20', '417', '436', '187', '268',
 '377', '75', '159', '60', '337', '148', '183', '299', '314', '449', '41', '46',
 '470', '43', '255', '295', '495', '42', '90', '496', '117', '169', '91', '188',
 '22', '394', '242', '429', '53', '315', '307', '333', '149', '105', '26', '110',
 '103', '411', '269', '79', '196', '208', '247', '190', '432', '238', '372',
 '339', '416', '424', '253', '223', '327', '489', '370', '178', '143', '152',
 '378', '35', '205', '335', '171', '232', '19', '199', '72', '80', '228', '439',
 '153', '345', '24', '32', '302', '445', '50', '425', '474', '330', '154', '292',
 '440', '444', '450', '132', '231', '252', '70', '38', '81', '83', '131', '37',
 '379', '334', '30', '47', '78', '365', '141', '200', '375', '475', '263', '346',
 '357', '33', '195', '464', '191', '469', '248', '390', '93', '59', '113', '322',
 '127', '155', '498', '36', '281', '338', '402', '396', '420', '55', '385',
 '387', '405', '363', '419', '356', '381', '126', '458', '175', '265', '348',
 '213', '284', '364', '490', '221', '170', '224', '151', '168', '306', '350',
 '48', '3', '67', '355', '493', '482', '297', '480', '481', '134', '368', '1',
 '102', '150', '162', '176', '397', '44', '305', '342', '94', '447', '225',
 '320', '266', '483', '235', '156', '275', '260', '262', '289', '164', '161',
 '181', '457', '56', '422', '215', '453', '116', '135', '216', '237', '172',
 '16', '194', '122', '211', '291', '96', '324', '472', '352', '492', '25', '51',
 '276', '336', '272', '240', '97', '298', '140', '177', '325', '184', '84',
 '106', '359', '101', '282', '88', '173', '6', '437', '139', '227', '418', '494',
 '374', '68', '167', '351', '400', '104', '414', '461', '109', '329', '349',
 '230', '77', '189', '229', '222', '10', '137', '202', '286', '12', '2', '404',
 '343', '344', '406', '294', '435', '160', '21', '15', '92', '236', '304', '380',
 '114', '193', '395', '144', '264', '258', '476', '409', '288', '115', '318',
 '5', '386', '69', '312', '309', '443', '499', '209', '256', '415', '433', '95',
 '233', '218', '186', '7', '112', '111', '362', '376', '251', '118', '241',
 '250', '87', '383', '421', '485', '430', '246', '463', '300', '212', '17',
 '426', '446', '311', '4', '478', '471', '319', '389', '331', '259', '323', '61',
 '455', '277', '468', '243', '354', '321', '384']

Instance tai079_100_10

100 jobs and 10 machines

NEH makespan with Taillard acceleration 6032.00

NEH verification with traditional method: 6032.00

Function 'neh' computation time: 0.0839s

Solution: ['12', '21', '13', '27', '62', '34', '77', '92', '70', '49', '81',
 '38', '88', '68', '15', '7', '31', '40', '29', '6', '54', '11', '46', '87',
 '71', '57', '44', '58', '26', '99', '30', '55', '60', '69', '19', '39', '50',
 '76', '14', '80', '24', '25', '78', '75', '43', '9', '4', '32', '56', '85',
 '36', '89', '93', '51', '10', '96', '16', '23', '79', '28', '41', '65', '67',
 '84', '82', '0', '95', '48', '83', '3', '73', '47', '64', '1', '61', '33', '66',
 '5', '59', '22', '98', '2', '52', '74', '17', '35', '37', '86', '97', '20',
 '18', '72', '8', '53', '94', '42', '91', '45', '63', '90']

Instance tai044_50_10
 50 jobs and 10 machines
 NEH makespan with Taillard acceleration 3198.00
 NEH verification with traditional method: 3198.00
 Function 'neh' computation time: 0.0099s
 Solution: ['46', '14', '42', '11', '25', '13', '3', '34', '1', '31', '37', '10',
 '0', '40', '20', '5', '22', '48', '29', '24', '35', '15', '39', '49', '47',
 '23', '27', '21', '16', '38', '33', '30', '2', '7', '44', '26', '36', '17', '6',
 '12', '45', '32', '28', '18', '9', '4', '43', '41', '8', '19']

Instance tai002_20_5
 20 jobs and 5 machines
 NEH makespan with Taillard acceleration 1365.00
 NEH verification with traditional method: 1365.00
 Function 'neh' computation time: 0.0034s
 Solution: ['12', '0', '4', '10', '6', '8', '1', '11', '7', '3', '19', '15',
 '14', '5', '16', '2', '17', '18', '9', '13']

Como podemos ver después de utilizar pypy para ejecutar el programa el tiempo de computación en los 3 primeros casos se reduce considerablemente:

Python3	pypy
5.3949s	0.5086s
0.1107s	0.0839s
0.0272s	0.0099s
0.0025s	0.0034s

2 Recent Applications of BRAs

2.1 Read in more detail than usual two BRA-related articles and write a half-a-page summary on each

2.1.1 A Discrete-Event Driven Metaheuristic For Dynamic Home Service Routing

El artículo analiza el uso de una metaheurística flexible dirigida por eventos discretos para facilitar las políticas de viajes compartidos en la industria de servicios para el hogar, lo que puede reducir la cantidad de vehículos y viajes requeridos. El enfoque está dirigido a escenarios dinámicos de enrutamiento y programación en tiempo real. En la experimentación se generan soluciones de manera rápida y eficiente, lo que lo hace adecuado para su uso en operaciones del mundo real. La metaheurística pretende ayudar con la complejidad de coordinar los tiempos de llegada de los miembros del personal y los vehículos en los lugares de recogida, y se puede utilizar para facilitar la reprogramación y el cambio de ruta según sea necesario.

El algoritmo propone un conjunto de agentes (A) que deben visitar y atender a un conjunto de clientes (N) mientras se adhieren a varias restricciones operativas, que incluyen ventanas de tiempo, niveles de calificación y regulaciones de tiempo de trabajo. Los agentes pueden viajar entre clientes

caminando o utilizando vehículos de compartidos (V). El objetivo es encontrar una solución de enrutamiento y programación que minimice el tiempo improductivo total, incluidos los tiempos de espera y los tiempos de viaje. Las variables de decisión se introducen para representar el modo de viaje de los agentes y los vehículos, también se definen una serie de restricciones para garantizar que los trabajos de los clientes se completen dentro de sus ventanas de tiempo, que cada cliente sea visitado por un agente calificado y hay tiempos para descanso. El problema maneja cancelaciones y nuevas solicitudes en tiempo real, lo que requiere reprogramación y desvío.

El algoritmo opera generando iterativamente soluciones prometedoras basadas en la ocurrencia de eventos a lo largo del tiempo. Estos eventos pueden estar relacionados con agentes (p. ej., próximo trabajo a realizar) o vehículos (p. ej., un agente debe ser recogido). El algoritmo utiliza técnicas de aleatorización sesgada (BRA) para variar las horas de inicio, las rutas a pie y los movimientos del personal para generar múltiples soluciones. La lista de eventos se ordena en orden cronológico según los tiempos de los eventos y se procesa iterativamente hasta que no quedan eventos. Se realizan verificaciones de factibilidad en todas las etapas del algoritmo y las soluciones no factibles se descartan y reemplazan con nuevas soluciones. El algoritmo se ejecuta en paralelo en múltiples núcleos para mejorar la eficiencia y es capaz de resolver de manera eficiente instancias del mundo real.

Evaluación El artículo me ha parecido muy interesante, creo que los temas que trata son bastante actuales y dada su alta relevancia las aplicaciones son muchas (*smart cities*, logística, sanidad, *car sharing*, etc.). Además, el artículo está bien estructurado y es muy fácil de entender. El rigor científico también destaca ya que presenta los problemas de manera clara y sencilla y los resuelve de forma exacta y concisa.

Criterio	Puntuación
Calidad del texto	9
Relevancia	8
Rigor	10
Aplicaciones	10
Total	9,25

2.1.2 *Biased-Randomized Iterated Local Search For A Multiperiod Vehicle Routing Problem With Price Discounts For Delivery Flexibility*

El problema de generación de rutas para vehículos de períodos múltiples (MPVRP) es una variante del problema de generación de rutas para vehículos (*Vehicle Routing Problem* (VRP)) en el que las demandas de los clientes deben cumplirse dentro de un período de tiempo específico, como una semana. Este problema se puede extender aún más al permitir flexibilidad en la entrega a cambio de un descuento en el precio si los clientes permiten que la entrega se realice un día antes o un día después. El objetivo de este artículo es determinar cuánto se pueden reducir los costes totales para diferentes niveles de flexibilidad de entrega y para diferentes niveles de descuentos de precios. Se propone un algoritmo de dos etapas que combina la búsqueda local iterada con técnicas de aleatorización sesgada. Este algoritmo funciona bien en una variedad de configuraciones, desde las más restringidas (los clientes solo pueden ser atendidos en su día preferido) hasta las más flexibles (los clientes pueden ser atendidos en cualquier día). También se analiza la relación entre la flexibilidad del cliente, los descuentos de precios y los costos totales y brindan información valiosa sobre los beneficios potenciales de la fijación de precios para la flexibilidad de entrega.

La descripción del problema se puede formular como un modelo donde la función objetivo minimiza el costo total de transporte para atender a todos los clientes y una serie de restricciones donde se asegura que cada cliente es visitado como máximo una vez o si se reparte el mismo día o que el vehículo salga del inicio una vez al día. Este modelo admite modificaciones, por ejemplo, el modelo se puede ampliar para permitir la entrega a un cliente en cualquier día excepto el día de entrega preferido, o cualquier día excepto los dos días antes y después del día de entrega preferido, etc. El modelo también se puede extender a considerar funciones objetivas adicionales, como minimizar el número de vehículos utilizados o el total distancia recorrida, o para considerar variables de decisión adicionales, como la asignación de clientes a los vehículos o la ruta de los vehículos.

Para resolver MRVRP con descuentos y flexibilidad de entrega, se desarrolla una metaheurística en dos etapas:

1. Se generan mapas de asignación de clientes a días prometedores, que se utilizan como soluciones iniciales para la segunda etapa. Esta etapa utiliza técnicas de aleatorización sesgada, que introducen aleatoriedad en el proceso de asignación de tal manera que las asignaciones más prometedoras reciben mayores probabilidades de ser seleccionadas.
2. Se refinan los costes de entrega de los mapas de asignación del cliente a los días más prometedores generados en la primera etapa. Esta etapa consiste en un proceso de búsqueda local iterada (ILS), que incluye una etapa de perturbación y una etapa de búsqueda local. En la etapa de perturbación, la asignación del cliente al día se perturba o modifica para escapar de los óptimos locales y explorar diferentes soluciones del espacio de soluciones. Se aplica un algoritmo de búsqueda local a la solución perturbada para mejorarla. Se repite ILS hasta que se alcanza la convergencia o un número máximo de iteraciones.

En general, el algoritmo BR-ILS genera múltiples mapas de asignación de cliente por día, estima sus costes de entrega usando una heurística de enrutamiento rápido y los refina para las asignaciones más prometedoras usando un proceso ILS. Esto permite que el algoritmo encuentre soluciones de mayor calidad para el MPVRP con descuentos por flexibilidad de entrega de una manera relativamente eficiente. El algoritmo también hace uso de varios parámetros, como el número máximo de iteraciones permitidas, el porcentaje del mapa base destruido en la fase de perturbación y el número de subconjuntos de mapas de mejor calidad (élite) que se seleccionarán de los mapas prometedores.

Evaluación Al igual que el artículo anterior, el tema tratado en este es también muy interesante, trata temas modernos y tiene muchas aplicaciones actuales. El texto es bastante claro y explícito y con pocos conocimientos matemáticos se puede comprender el algoritmo descrito. Creo que el artículo presenta al lector una visión bastante amplia de los temas relacionados con BRA y VRP de forma rigurosa pero amena acercándolo a un problema real de gestión y logística.

Criterio	Puntuación
Calidad del texto	8
Relevancia	10
Rigor	10
Aplicaciones	10
Total	9,5

3 *Simheuristics Theory and Practice*

3.1 *A Simheuristic For Routing Electric Vehicles With Limited Driving Ranges And Stochastic Travel Times*

Este artículo presenta un algoritmo simheurístico para resolver un problema de enrutamiento de vehículos en el que se consideran tanto las restricciones del rango de conducción como los tiempos de viaje estocásticos. El objetivo es minimizar el costo esperado basado en el tiempo requerido para completar el plan de distribución de carga. La simheurística combina la simulación de Monte Carlo con una metaheurística de inicio múltiple que utiliza técnicas de aleatorización sesgada. Se llevan a cabo experimentos computacionales para probar la efectividad de la simheurística y analizar el impacto de la incertidumbre en los planes de enrutamiento. La simheurística pretende extender las capacidades de las metaheurísticas para tratar problemas estocásticos en el contexto del transporte verde y las ciudades inteligentes, donde el uso de vehículos eléctricos es cada vez más importante para la sostenibilidad.

En las ciudades inteligentes es necesario gestionar los recursos de forma inteligente para lograr un crecimiento sostenible y al mismo tiempo satisfacer las demandas de los consumidores, que requieren una intensa actividad de transporte de mercancías. No obstante, esta actividad también debe realizarse sin generar ineficiencias económicas o impactos negativos sobre el medio ambiente o los ciudadanos. Para abordar estos desafíos, se han implementado iniciativas y políticas para promover el uso de vehículos eléctricos (EV) en las actividades de transporte y logística de la ciudad, ya que los EV tienen el potencial de reducir las externalidades en el medio ambiente y los ciudadanos. Los vehículos eléctricos tienen restricciones de autonomía y sus tasas de consumo de batería se ven afectadas por una variedad de factores impredecibles, que incluyen la congestión del tráfico, las condiciones climáticas y el comportamiento del conductor. En el artículo se abordan estos desafíos mediante el análisis de un problema realista de enrutamiento de vehículos que considera las restricciones del rango de conducción y los tiempos de viaje estocásticos, con el objetivo de minimizar el costo esperado basado en el tiempo requerido para completar el plan de distribución de carga.

El problema se define en un gráfico con un depósito (nodo 0) y un conjunto de clientes, y cada cliente tiene una demanda que debe ser satisfecha por un conjunto de vehículos con una determinada capacidad de carga. El costo de viajar a lo largo de cada borde viene dado por una variable aleatoria que sigue una distribución de probabilidad conocida con un valor medio. El objetivo es minimizar el costo esperado basado en el tiempo de completar el proceso de entrega al mismo tiempo que se considera la restricción de que el tiempo de viaje esperado de cada vehículo debe estar limitado por la duración de su batería. Sin embargo, considerar los tiempos de viaje estocásticos introduce incertidumbre sobre cuánta energía se requerirá para completar una ruta, lo que puede dificultar la garantía de soluciones factibles cuando existen fuertes restricciones en la duración de la batería. Se propone un algoritmo simheurístico para resolver un problema de enrutamiento de vehículos que tiene en cuenta las restricciones del campo de prácticas y los tiempos de viaje estocásticos. La simheurística combina la simulación de Monte Carlo (MCS) con una metaheurística de inicio múltiple aleatorizado sesgado (BR-MS). La metaheurística BR-MS genera y mejora soluciones utilizando una versión aleatoria sesgada de la búsqueda local y heurística de Clarke and Wright Savings, mientras que el componente MCS estima el costo esperado basado en el tiempo de las soluciones en un entorno estocástico. El algoritmo procede a través de un proceso de inicio múltiple, en el que se generan una serie de soluciones deterministas, se mejoran con la búsqueda local y se evalúan con un MCS rápido. Las mejores soluciones se evalúan luego con un MCS más intensivo.

La complejidad computacional del algoritmo es $O(\maxTime)$, y los autores realizan experimentos computacionales para probar su desempeño y analizar el efecto de la incertidumbre en los planes de enrutamiento.

Evaluación El artículo trata temas muy relevantes para el futuro próximo y aborda los problemas que pueden surgir de forma muy amena y clara. El algoritmo y el análisis del mismo se explica con mucha claridad lo cual hace que el texto se entienda de forma muy completa.

Criterio	Puntuación
Calidad del texto	9
Relevancia	9
Rigor	10
Aplicaciones	9
Total	9,25

3.2 *Fuzzy Simheuristics for Optimizing Transportation Systems: Dealing with Stochastic and Fuzzy Uncertainty*

El enfoque simheurístico permite el modelado de diferentes componentes, como tiempos de viaje, tiempos de servicio y demandas de los clientes, como deterministas, estocásticos o difusos, y se puede aplicar a problemas que incluyen el problema de enrutamiento de vehículos, el problema de enrutamiento de arco y el equipo. problema de orientación. Los autores realizan experimentos computacionales para validar la eficacia del enfoque de simheurística y sugieren que también se puede extender a otros problemas de optimización en áreas como la fabricación, la producción, las ciudades inteligentes y las redes de telecomunicaciones.

En este artículo, los autores proponen un enfoque simheurístico difuso para resolver tanto VRP como el problema de orientación en equipo (TOP) bajo escenarios de incertidumbre que incluyen incertidumbre tanto probabilística como no probabilística. El enfoque simheurístico difuso combina una metaheurística de inicio múltiple con sistemas de simulación de Monte Carlo e inferencia difusa para manejar variables estocásticas y difusas, respectivamente. El método de solución consta de tres etapas:

1. construcción de una solución factible inicial a través de una heurística constructiva basada en el ahorro
2. la generación de soluciones múltiples a través de decisiones aleatorias sesgadas incorporadas en un marco de inicio múltiple
3. refinamiento en la que un número mayor de ejecuciones de simulación se aplican a un conjunto de soluciones de élite.

Los resultados de los experimentos computacionales que evalúan el desempeño del enfoque simheurístico difuso para resolver el problema de enrutamiento de vehículos (VRP) y el problema de orientación en equipo (TOP) en condiciones de incertidumbre muestran que el enfoque simheurístico difuso puede proporcionar soluciones casi óptimas para ambos problemas en condiciones de incertidumbre. Los resultados también muestran que las soluciones obtenidas utilizando el enfoque simheurístico difuso superan a las soluciones OBD cuando se simulan en condiciones estocásticas, con una mejora promedio de 7.91% para el VRP y 1.72% para el TOP. Por último, estos resultados sugieren que el uso de simheurísticas difusas podría extenderse a otros problemas

de optimización en la fabricación, la producción, las ciudades inteligentes y las redes de telecomunicaciones. El trabajo futuro podría implicar la adaptación del enfoque a versiones más ricas de VRP y TOP que incluyen múltiples depósitos, ubicación de instalaciones, ventanas de tiempo y condiciones externas dinámicas.

Evaluación Igual que en artículo anterior, artículo trata temas de considerable relevancia para el futuro próximo y aborda los problemas que pueden surgir, aunque no de una forma tan clara. Creo que el concepto de lógica difusa no queda muy claro y se combinan muchas disciplinas y conceptos de forma que el artículo queda muy completo pero un poco difícil de entender

Criterio	Puntuación
Calidad del texto	9
Relevancia	9
Rigor	10
Aplicaciones	8
Total	9

4 *Learnheuristics and Agile Optimization*

4.1 *Read at least two Learnheuristics-related articles and write a brief summary on them.*

4.1.1 *A Biased-Randomized Learnheuristic for Solving the Team Orienteering Problem with Dynamic Rewards*

En este artículo, se discute el problema de orientación en equipo (TOP) con entradas dinámicas. En la versión dinámica del problema, la recompensa asociada a cada nodo puede variar según el orden en que se visite. Para resolver este problema, se propone una heurística de aprendizaje (BR-LH) que combina un enfoque basado en heurística con un mecanismo de “aprendizaje” para estimar las recompensas reales a medida que se toman nuevas decisiones.

El problema de orientación en equipo (TOP) implica diseñar rutas para una flota de vehículos para maximizar la recompensa total recolectada, dado un umbral de tiempo limitado. La flota consta de m vehículos, y el conjunto de posibles nodos a visitar se puede describir mediante un grafo no dirigido G con un conjunto de n nodos N y un conjunto de aristas E que conectan los nodos. La recompensa por visitar un nodo es u_i , y el tiempo de viaje para atravesar un borde (i, j) es t_{ij} . El objetivo es encontrar un conjunto M de m rutas, donde cada ruta está definida por una matriz de nodos que comienzan en el nodo 1 y llegan al nodo n , de modo que la suma de las recompensas recolectadas se maximice sin exceder un umbral de tiempo t_{max} para cada ruta. Las restricciones para el problema incluyen un umbral de tiempo para completar cada ruta, un límite de una visita a cada nodo y el requisito de que cada ruta comience en el depósito de origen y finalice en el depósito final.

La heurística aleatoria (BR-H) implica construir una solución inicial mediante la construcción de una ruta que conecta cada nodo de cliente con los depósitos de origen y final, y luego usar un concepto de “ahorro” y “preferencia” para determinar el orden en que se fusionan las rutas. El concepto de preferencia es una combinación lineal de ahorros basados en el tiempo y recompensas acumuladas, y se utiliza para generar una lista ordenada de fusiones de rutas potenciales. Estas

fusiones se completan en orden de preferencia, pero solo si el tiempo total esperado después de la operación no supera el umbral de tiempo máximo. El BR-H se transforma en un algoritmo probabilístico mediante el uso de una distribución geométrica para impulsar la selección del siguiente elemento de la lista de ahorro y encapsular estos pasos en un procedimiento de inicio múltiple.

En la evaluación experimental, el algoritmo BR-LH pudo superar al algoritmo BR-H en un conjunto de instancias de referencia para el TOP dinámico, con una mejora promedio de alrededor del 4,9%. Los resultados también mostraron que el uso de un método estático para la versión dinámica del problema conduce a soluciones subóptimas. El trabajo futuro en el algoritmo BR-LH incluye mejorar el mecanismo de aprendizaje y probar el algoritmo en instancias más grandes y con diferentes componentes dinámicos.

4.1.2 Evaluación

El artículo propone enfoques modernos a problemas de disciplinas también actuales (orientación en equipo dinámico). Es interesante ver cómo distintas heurísticas se combinan para resolver el problema y enfocar el problema desde un punto de vista diferente. El texto es fácil y comprensible y está explicado de una forma muy didáctica que me ha hecho entender los conceptos y qué relación hay entre ellos.

Criterio	Puntuación
Calidad del texto	10
Relevancia	9
Rigor	10
Aplicaciones	9
Total	9,5

4.1.3 *A Simheuristic-Learnheuristic Algorithm For The Stochastic Team Orienteering Problem With Dynamic Rewards*

En este trabajo se propone un enfoque heurístico de aprendizaje para el STOPDR. El enfoque es una combinación de una heurística aleatoria sesgada con un mecanismo de aprendizaje que estima las verdaderas recompensas a medida que se toman nuevas decisiones. La heurística aleatoria sesgada (BR-H) se basa en el concepto de “ahorro” y “preferencia” para determinar el orden en que se fusionan las rutas. La preferencia es una combinación lineal de ahorros basados en el tiempo y recompensas acumuladas, y se utiliza para generar una lista ordenada de posibles operaciones de combinación de rutas. Estas fusiones se completan en orden de preferencia, pero solo si el tiempo total esperado después de la operación no supera el límite de tiempo máximo. El BR-H se transforma en un algoritmo probabilístico utilizando una distribución geométrica para impulsar la selección del siguiente elemento en la lista de ahorro y encapsulando estos pasos en un procedimiento de inicio múltiple.

En cada paso de la heurística constructiva encapsulada en el algoritmo BR-H. El componente dinámico puede incluir bonificaciones por visitar nodos al principio de una ruta y penalizaciones por nodos visitados al final de la ruta. Los autores proponen un algoritmo heurístico de aprendizaje de aleatorización sesgada (BRLH) para resolver el STOPDR, que combina un algoritmo de aleatorización sesgada con simulación y aprendizaje computacional. El BRLH puede aprender las recompensas dinámicas a través de las observaciones obtenidas de la simulación y utiliza este aprendizaje para mejorar el rendimiento del algoritmo de aleatorización sesgada.

El problema de orientación en equipo (TOP) es un tipo de problema de optimización combinatoria que involucra el diseño de rutas para una flota de vehículos para maximizar la recompensa total recolectada dentro de un período de tiempo limitado. La flota consta de m vehículos y el conjunto de posibles nodos a visitar se puede describir mediante un grafo no dirigido G con un conjunto de n nodos N y un conjunto de aristas E que conectan los nodos. La recompensa por visitar un nodo es u_i , y el tiempo de viaje para atravesar un borde (i, j) es t_{ij} . El objetivo es encontrar un conjunto M de m rutas, donde cada ruta está definida por una matriz de nodos que comienzan en el nodo 1 y terminan en el nodo n , de modo que la suma de las recompensas recolectadas se maximice sin exceder un umbral de tiempo t_{max} para cada ruta. Las restricciones del problema incluyen un umbral de tiempo para completar cada ruta, un límite de una visita a cada nodo y el requisito de que cada ruta comience en el depósito de origen y finalice en el depósito final.

El problema se vuelve más difícil cuando las recompensas asociadas con cada nodo son dinámicas, lo que significa que pueden cambiar según el orden en que se visitan. Esto se conoce como el problema de la orientación en equipo con recompensas dinámicas (STOPDR). En STOPDR, puede haber bonificaciones por visitar nodos al principio de una ruta y penalizaciones por visitar nodos al final de una ruta. Estos valores de bonificación y penalización son desconocidos y deben aprenderse de las observaciones de la simulación o de una simulación detallada del proceso.

El algoritmo BRLH se prueba en un conjunto de instancias basadas en las instancias de referencia clásicas para el TOP estático, y se demuestra que puede aprender y explotar con éxito información sobre bonificaciones y penalizaciones, así como lograr recompensas promedio altas y buenos niveles de fiabilidad.

4.1.4 Evaluación

En primer lugar, es importante destacar la relevancia del tema del artículo. Los enfoques heurísticos de aprendizaje automático son un área de investigación activa y en constante evolución, y su aplicación en la resolución de problemas es esencial en muchas áreas de la industria y la ciencia.

El artículo parece estar bien fundamentado y basado en la literatura existente sobre el tema. Además, se presentan resultados de experimentos y se comparan con otros métodos para evaluar la efectividad de los enfoques heurísticos.

Los enfoques heurísticos de aprendizaje automático tienen una amplia gama de aplicaciones prácticas, y el artículo hace un buen trabajo al mencionar ejemplos concretos y discutir cómo estos enfoques se han utilizado en la industria. Además, se discute cómo estos enfoques pueden ser útiles en una variedad de problemas y se presentan algunas ideas sobre cómo podrían ser aplicados en el futuro.

Criterio	Puntuación
Calidad del texto	10
Relevancia	9
Rigor	10
Aplicaciones	9
Total	9,5

4.2 *Read at least two Agile-Optimization related articles and write a brief summary on them.*

4.2.1 *Agile Optimization Of A Two-Echelon Vehicle Routing Problem With Pickup And Delivery*

En este artículo, se presenta un problema de enrutamiento de vehículos en el que una flota de vehículos debe entregar materias primas o productos a granel a nodos intermedios, recolectar los artículos procesados solicitados por los clientes y entregarlos a los clientes finales antes de regresar al depósito. El problema debe resolverse en tiempo real y se propone una heurística constructiva original para proporcionar una solución factible y razonablemente buena. La heurística se extiende a un algoritmo aleatorizado sesgado que puede generar resultados aún mejores en ejecuciones paralelas. Los resultados muestran que la metodología propuesta genera resultados competitivos en milisegundos y supera a otras heurísticas de la literatura.

Los autores proponen un método de solución para un problema de optimización en tiempo real relacionado con la distribución de mercancías mediante drones en situaciones de desastre. El problema es una variante del problema de enrutamiento de vehículos (VRP), llamado VRP de dos escalones (2E-VRP), en el que los bienes se entregan desde un almacén central a las instalaciones intermedias, y luego desde las instalaciones intermedias a los clientes finales. Para resolver este problema en tiempo real los autores proponen una heurística constructiva rápida que luego se extiende a un algoritmo aleatorio sesgado (BR). El algoritmo BR es capaz de proporcionar “buenas” soluciones a instancias medianas y grandes en milisegundos, utilizando computación paralela. Los resultados del método de solución propuesto se comparan con los de otras heurísticas de la literatura y muestran que es capaz de generar soluciones competitivas en tiempo real.

La solución pasa por crear una heurística constructiva rápida llamada LH, que luego se extiende a un algoritmo aleatorio sesgado (BR). La hibridación de la heurística BR y la computación paralela conduce al concepto de “optimización ágil”, que implica la paralelización masiva de algoritmos BR para encontrar buenas soluciones para problemas de optimización NP-difíciles y a gran escala en tiempo real. La heurística LH consta de cuatro etapas:

1. Crear una solución ficticia con rutas que sirven a nodos individuales.
2. Fusionar las rutas iniciales usando una heurística constructiva.
3. Mejorar las rutas fusionadas usando una búsqueda local de 2 opciones
4. Reasignar las rutas a los drones utilizando una heurística constructiva.

El algoritmo BR implica ejecutar la heurística LH varias veces en paralelo, utilizando una distribución de probabilidad sesgada para modificar su comportamiento codicioso y potencialmente encontrar soluciones aún mejores.

Los resultados muestran que la metodología propuesta genera resultados competitivos en milisegundos, y es capaz de superar otras heurísticas en algunos casos. Los autores también aplican el enfoque a un escenario del mundo real de distribución de drogas después de un desastre y demuestran su eficacia para brindar soluciones rápidas y factibles.

Evaluación Este artículo presenta una solución original al problema de enrutamiento de vehículos. En mi opinión, la manera de generar una solución mediante la solución ficticia es una forma muy original de atajar este problema. Además, el artículo habla de temas muy presentes ahora como el enrutado de drones y siempre desde el marco de las soluciones en tiempo real. En definitiva, es un artículo con una solución muy original y moderna para problemas modernos.

Criterio	Puntuación
Calidad del texto	10
Relevancia	10
Rigor	10
Aplicaciones	9
Total	9,75

4.2.2 *Optimizing Ride-Sharing Operations in Smart Sustainable Cities: Challenges and the Need for Agile Algorithms*

Las actividades de transporte y logística (T&L), como los viajes compartidos son importantes para mejorar la eficiencia y la sostenibilidad del transporte en las zonas urbanas. Estas actividades pueden ayudar a reducir la congestión del tráfico y la contaminación, pero también enfrentan desafíos que incluyen regulación, seguridad y convivencia con los servicios de transporte tradicionales. Los avances en las tecnologías de la información y la comunicación han hecho posible el desarrollo de modos de transporte más inteligentes y sostenibles.

Los problemas de viajes compartidos suelen ser a gran escala y NP-difíciles, lo que dificulta el uso de métodos de solución exactos para resolverlos de manera óptima. Las metaheurísticas, que son métodos aproximados que pueden encontrar buenas soluciones rápidamente, a menudo se usan para resolver problemas de viajes compartidos, pero pueden no ser adecuadas para situaciones en tiempo real. Los algoritmos de optimización ágil (AO) son una nueva herramienta para resolver problemas de optimización en tiempo real mediante la paralelización masiva de algoritmos aleatorizados sesgados. Los algoritmos de AO son flexibles, rápidos y requieren el ajuste de unos pocos parámetros, y se pueden usar para resolver problemas dinámicos de viajes compartidos mediante la reoptimización continua del sistema a medida que se incorporan nuevos datos. Los algoritmos de AO se pueden hibridar con otras técnicas, como la simulación o el aprendizaje automático, para mejorar aún más su eficacia. Los algoritmos de AO tienen el potencial de mejorar significativamente la eficiencia y la sostenibilidad de los sistemas de viajes compartidos.

Los viajes compartidos y los vehículos compartidos pueden tener beneficios tales como costos reducidos y consumo de energía, y niveles más bajos de contaminación en áreas urbanas. Sin embargo, los formuladores de políticas enfrentan desafíos en el diseño, desarrollo e implementación de estos sistemas en la vida real, incluidos los problemas de seguridad y regulación. La literatura existente sobre la optimización de viajes compartidos y vehículos compartidos puede ayudar a identificar problemas y analizar alternativas basadas en experiencias en otras áreas urbanas, y puede clasificarse por metodología analítica, incluidos métodos exactos, metaheurísticas y simulación. Los desafíos y las oportunidades de investigación para optimizar los viajes compartidos incluyen problemas de sincronización y coordinación, la inclusión de vehículos eléctricos y autónomos, y factores como flotas de vehículos heterogéneas y condiciones dinámicas e inciertas. Los enfoques híbridos que combinan metaheurísticas con métodos de simulación, aprendizaje automático y algoritmos de optimización ágiles que pueden generar buenas soluciones en tiempo real y útiles para abordar estos desafíos y oportunidades.

Evaluación El artículo presenta una situación muy amplia del problema de *car sharing* y una explicación muy exhaustiva de las distintas soluciones a estos problemas mediante distintas técnicas y algoritmos. Quizá es demasiado extensivo y podría plantearse más como un survey. Sin embargo,

el artículo es demasiado extenso y la explicación sobre algoritmos ágiles no es demasiado exhaustiva, aunque quizá este sea el objetivo del artículo

Criterio	Puntuación
Calidad del texto	7
Relevancia	10
Rigor	8
Aplicaciones	8
Total	8,25