Michael Smurfit Business School
Blackrock, May - July 2021

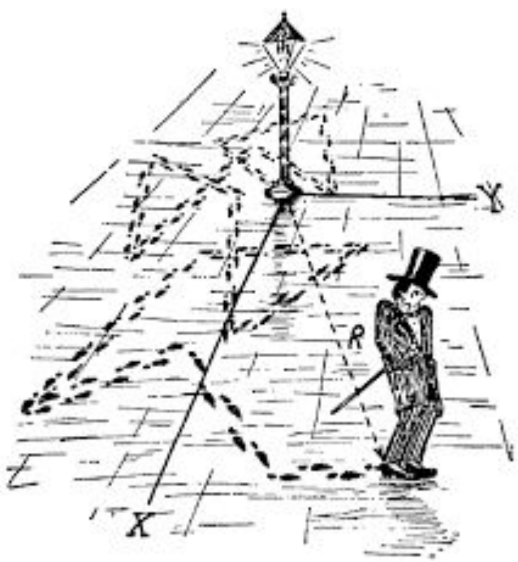
Random Search & CWS Heuristic VRP

Prof. Dr. Angel A. Juan
ajuanp@gmail.com | http://ajuanp.wordpress.com
IN3 - Computer Science Dept., UOC, Barcelona, Spain

UOC R&I IN3 Internet Interdisciplinary Institute
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA
University College Dublin Ireland's Global University
Universitat Oberta de Catalunya
UAB Universitat Autònoma de Barcelona
UNIVERSIDADE ABERTA www.uab.pt
Euncet Business School

Overview

- Part I: Random Search Concepts
- Part II: Random Search (BFP) w. Python
- Part III: CW Savings Heuristic (VRP) w. Python
- References

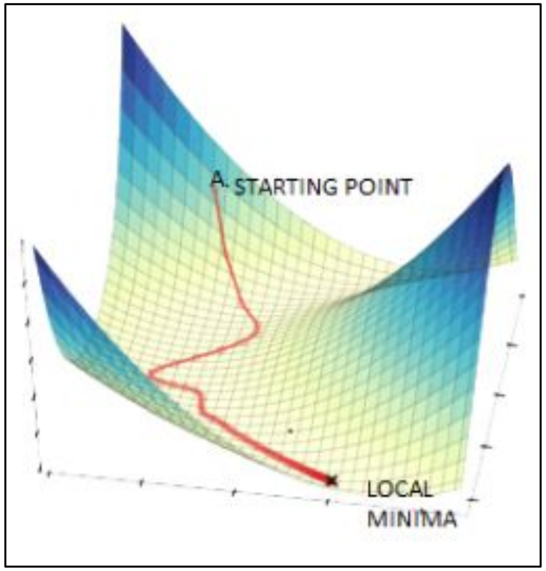


Part I:

Random Search Concepts

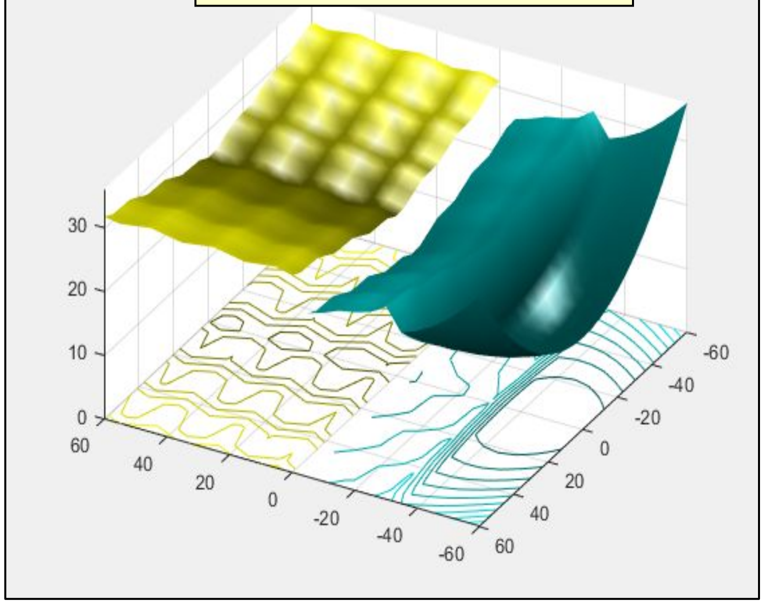
Random Search (RS) Basic Concepts

- Random search (RS) is a family of numerical optimization methods that make use of randomness instead of using the gradient of the objective function, i.e., they are derivative-free methods.
- Hence, RS can be used on non-smooth objective functions.
- The strategy is to sample solutions from across the entire search space, typically using a uniform probability distribution.
- Biased-Randomized Algorithms (BRAs) are a particular case of RS, in which the probability distribution is skewed instead of uniform.



Gradient-descent search

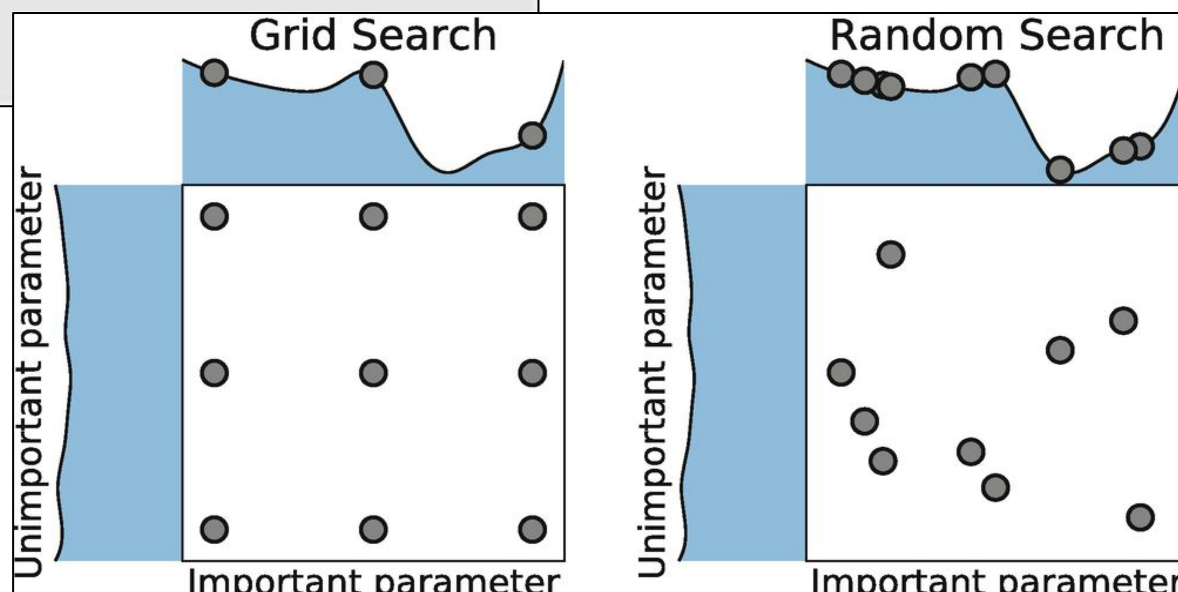
Non-smooth objective function



Pseudocode of Random Search

PROCEDURE pure random search()
    InputInstance();
    Set $y = -\infty$;
    DO
        Generate a point x from the uniform distribution over S;
        Set $y = \max(y, f(x))$;
    OD;
    RETURN(y);
END pure random search;

Simple random search for a maximization problem



Part II:

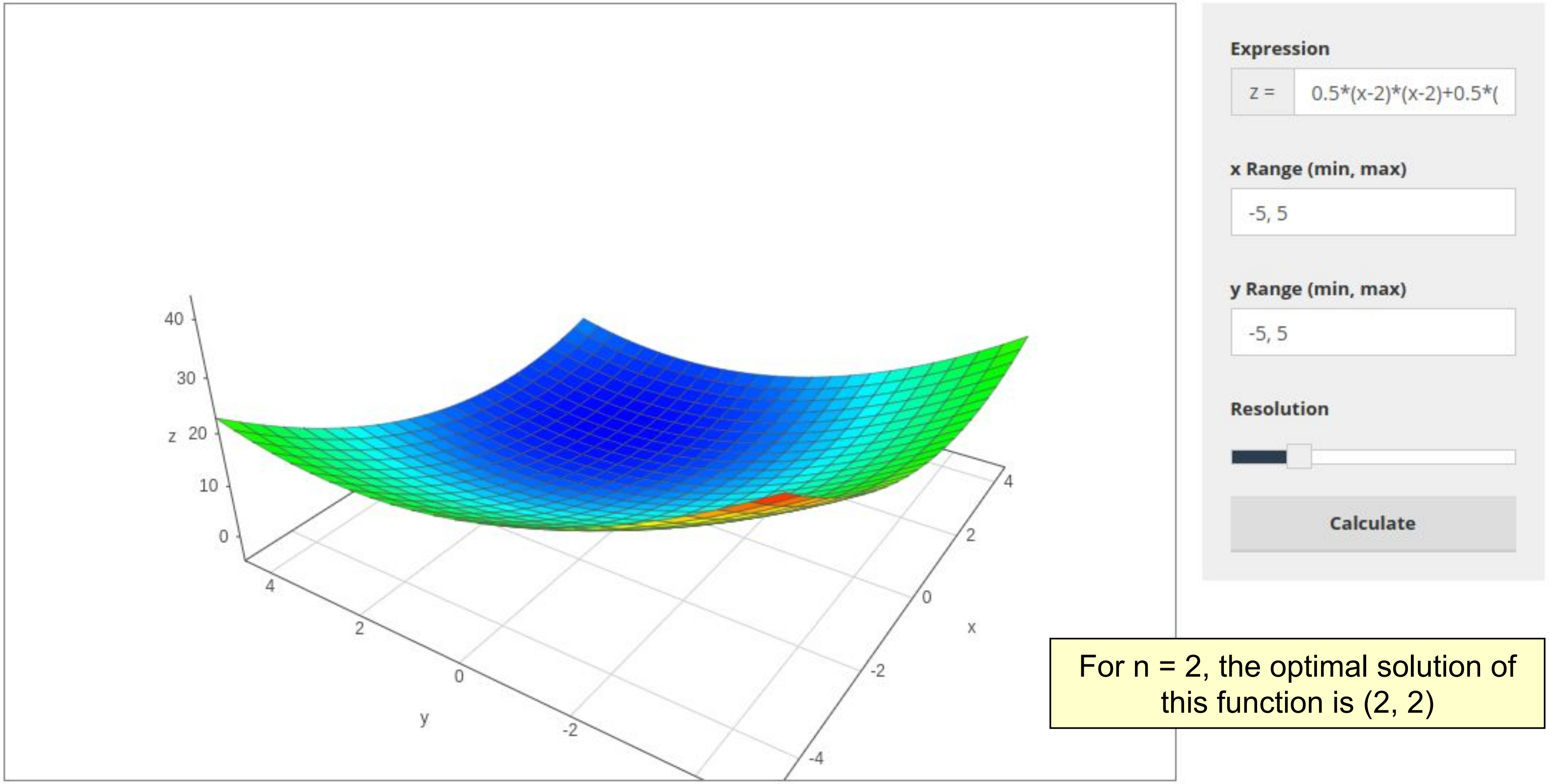
Random Search (BFP) w. Python

The Basin Function Problem (BFP)

It is an instance of a continuous function optimization that seeks to minimize f(x) where:

$$f(x) = \sum_{i=1}^{n} a(x_i - h)^2 + k$$

$$\forall -5.0 < x_i < 5.0 \text{ and } n = 2, a = 0.5, h = 2, k = -5$$



For n = 2, the optimal solution of this function is (2, 2)

Plotting Surfaces Online

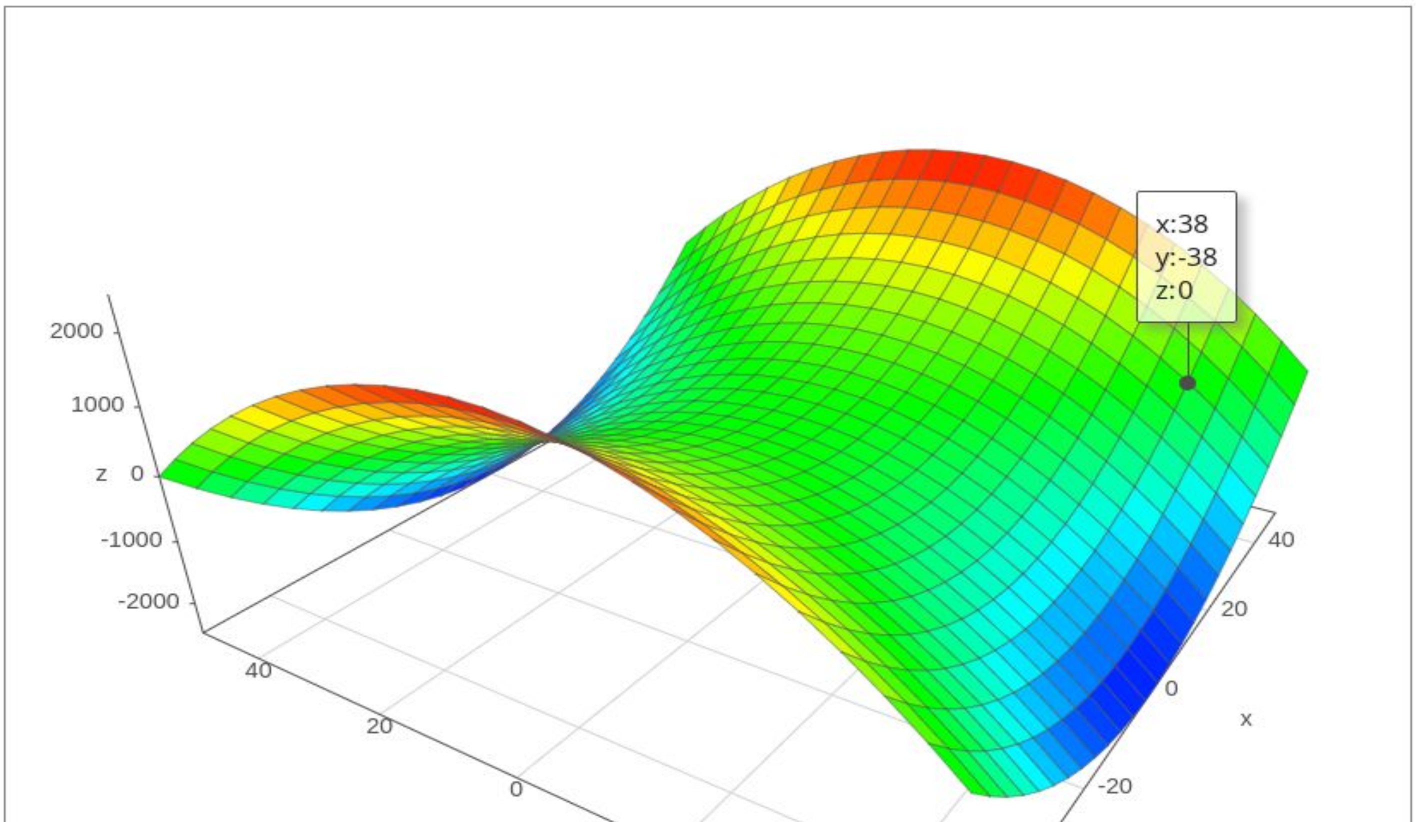
3D Surface Plotter

https://academo.org/demos/3d-surface-plotter/

An online tool to create 3D plots of surfaces.

Maths Geometry Graph Plot Surface

Share Tweet



Expression

z = x*x - y*y

x Range (min, max)

-50, 50

y Range (min, max)

-50, 50

Resolution

Calculate

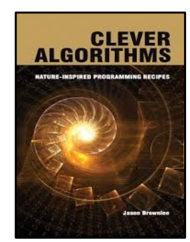
Shared.py   Basin Function

```python
import math, random


'''
AUXILIARY CODE FOR BASIN FUNCTION
'''
# An adapted basin function is used as described in:
#    http://www.saipanyam.net/2011/06/stochastic-algorithms-1.html
def basinFunction(vector):
    a, h, k = 0.5, 2, -5
    sum = 0
    for item in vector:
        sum = sum + a * pow((item - h), 2) + k
    return sum

```

spyder

RS_BF.py Generation of a Random Solution

```python
from Shared import basinFunction
import random, time

# Select random values inside search space for solution vector (x1, ..., xn)
def randomSolution(searchSpace, problemSize):
    min = searchSpace[0] # min value in search space
    max = searchSpace[1] # max value in search space
    inputValues = [] # list to store xi values
    for i in range(0, problemSize):
        # generate values xi between the min and max values
        inputValues.append(min + (max - min) * random.random())
    return inputValues
```



Python code on RS is based on the ones by Jason Brownlee (http://www.cleveralgorithms.com/) and Sain Panyam (https://www.saipanyam.net/2011/06/clever-algorithms-python.html)

RS_BF.py   Algorithm Framework

```python
''' ALGORITHM FRAMEWORK '''
# Random Search algorithm to optimize the adapted basin function
algorithmName = "Random Search"
searchSpace = [-5, 5] # Interval for input xi
maxIterations = 10000
problemSize = 2 # dimension n of the sol vector (x1, ..., xn)
print("Best Sol by " + algorithmName + "...")
start = time.clock()
bestCost = float("inf") # infinity
while maxIterations > 0:
    maxIterations -= 1
    newSol = randomSolution(searchSpace, problemSize)
    newCost = basinFunction(newSol)
    # if newCost is better than bestCost then update bestSol
    if newCost < bestCost:
        bestSol = newSol
        bestCost = newCost
stop = time.clock()
print("Cost = ", bestCost)
print("Sol = ", bestSol)
print("Elapsed = ", stop - start)
```

RS_BF.py Results After 10000 Iterations (1 run)

IPython console

Console 1/A

```
In [1]: runfile('/home/aajp/Documents/Intelligent Algorithms Python/
RS_BF.py', wdir='/home/aajp/Documents/Intelligent Algorithms Python')
Best Sol by Random Search...
Cost =  -9.998196923023961
Sol =  [2.042832823498701, 2.0420892288264483]
Elapsed =  0.015645000000000002
```

This is a probabilistic algorithm --> using more iterations (or more runs in parallel) will help to get better solutions.

spyder

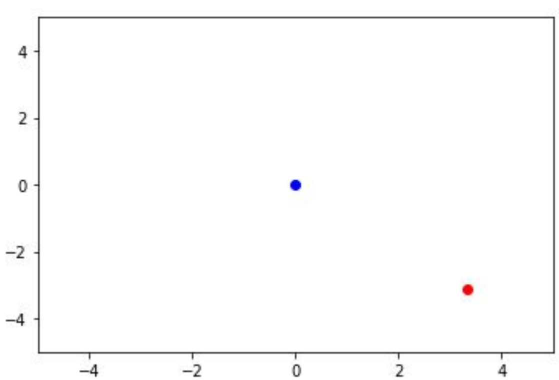
RS_BF.py Visualizing the Random Search

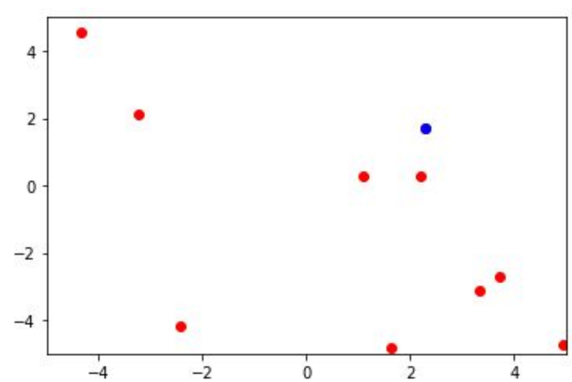
```python
import matplotlib.pyplot as plt
```

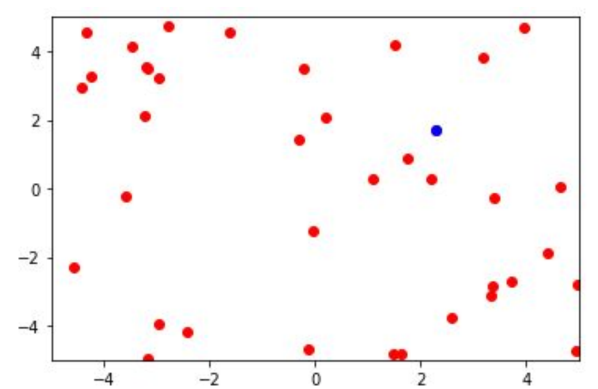
```python
maxIterations = 100
```

Visualization significantly increases computational time and required resources, so it is better to set a low number of iterations.

```python
x = []
y = []
bestSol = [0,0]
while maxIterations > 0:
    maxIterations -= 1
    plt.axis([-5,5,-5,5])
    newSol = randomSolution(searchSpace, problemSize)
    newCost = basinFunction(newSol)
    x.append(newSol[0])
    y.append(newSol[1])
    plt.scatter(x, y, color='r');
    plt.scatter(bestSol[0],bestSol[1], color='b');
    plt.show()
    if newCost < bestCost:
        bestSol = newSol
        bestCost = newCost
```

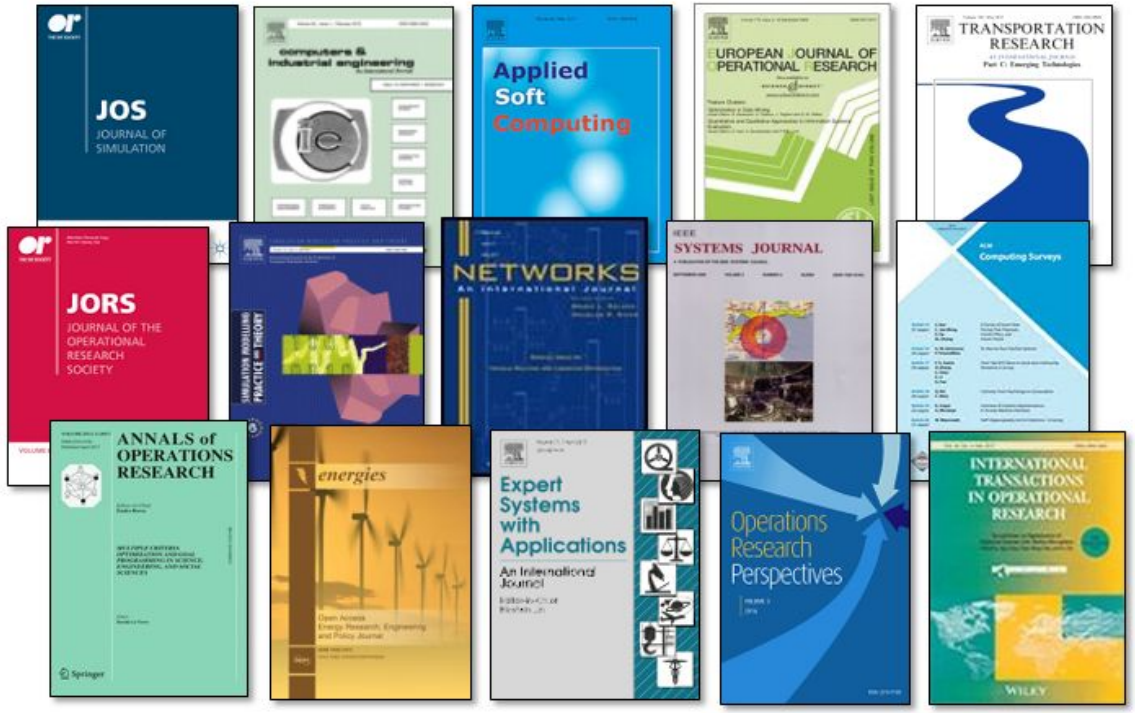






References

Faulin, J., Gilibert, M., Juan, A. A., Vilajosana, X., & Ruiz, R. (2008). SR-1: A simulation-based algorithm for the capacitated vehicle routing problem. In 2008 Winter Simulation Conference (pp. 2708-2716). IEEE.



1

2

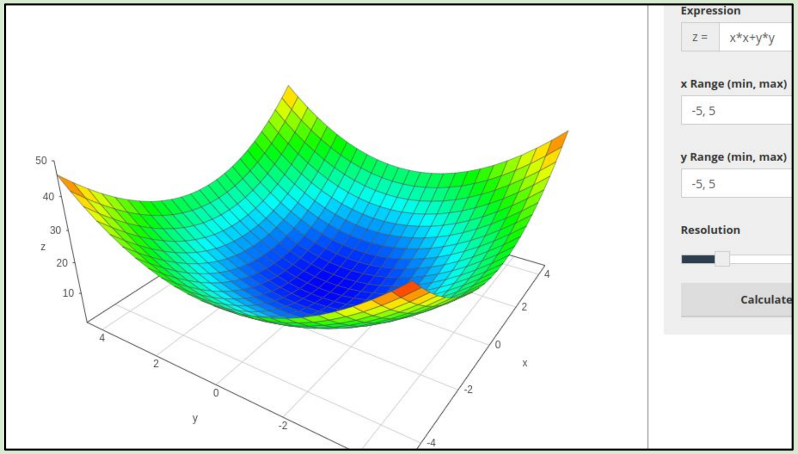
https://informs-sim.org/

Winter Simulation Conference Archive

https://www.dropbox.com/sh/qfk6i2858v8dz2w/AADCeHaLPIEpFOQJPdfQNbwha?dl=0

Homework Activities

1. Construct your own **Python program** to implement a RS algorithm for solving the BFP.



2. Solve the same problem but for the following basin function:

It is an instance of a continuous function optimization that seeks to minimize f(x) where:

$$f(x) = \sum_{i=1}^{n} x_i^2$$

$$\forall -5.0 < x_i < 5.0 \; and \; n = 2$$

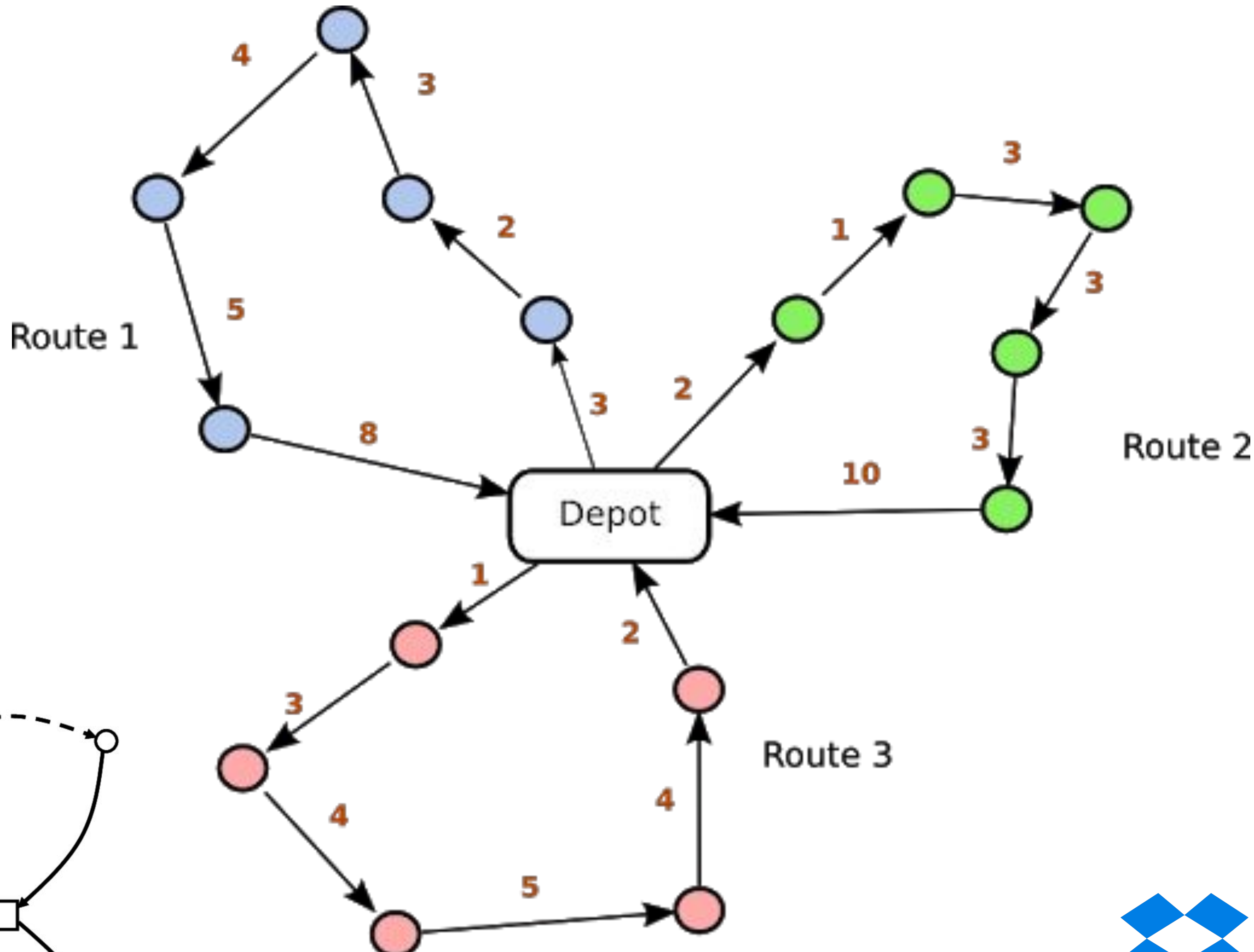
3. Read the **SR1 article** (https://www.informs-sim.org/wsc08papers/341.pdf) from the WSC'08 and write a brief summary on it. Assign a score between 0 and 10.

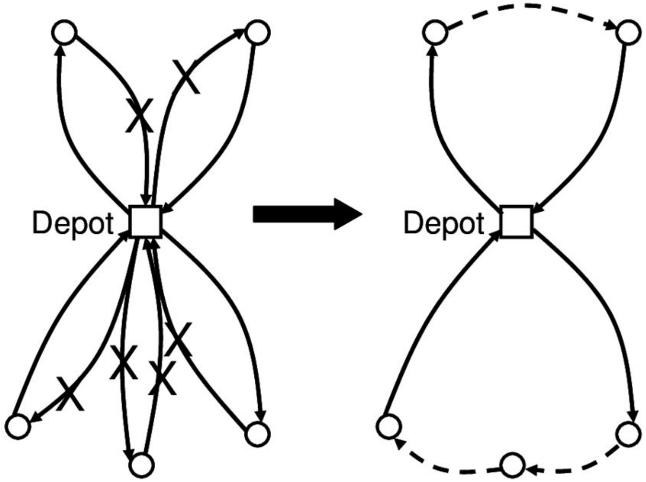
Part III:

CW Savings Heuristic (VRP) w. Python

The Savings Heuristic for the VRP

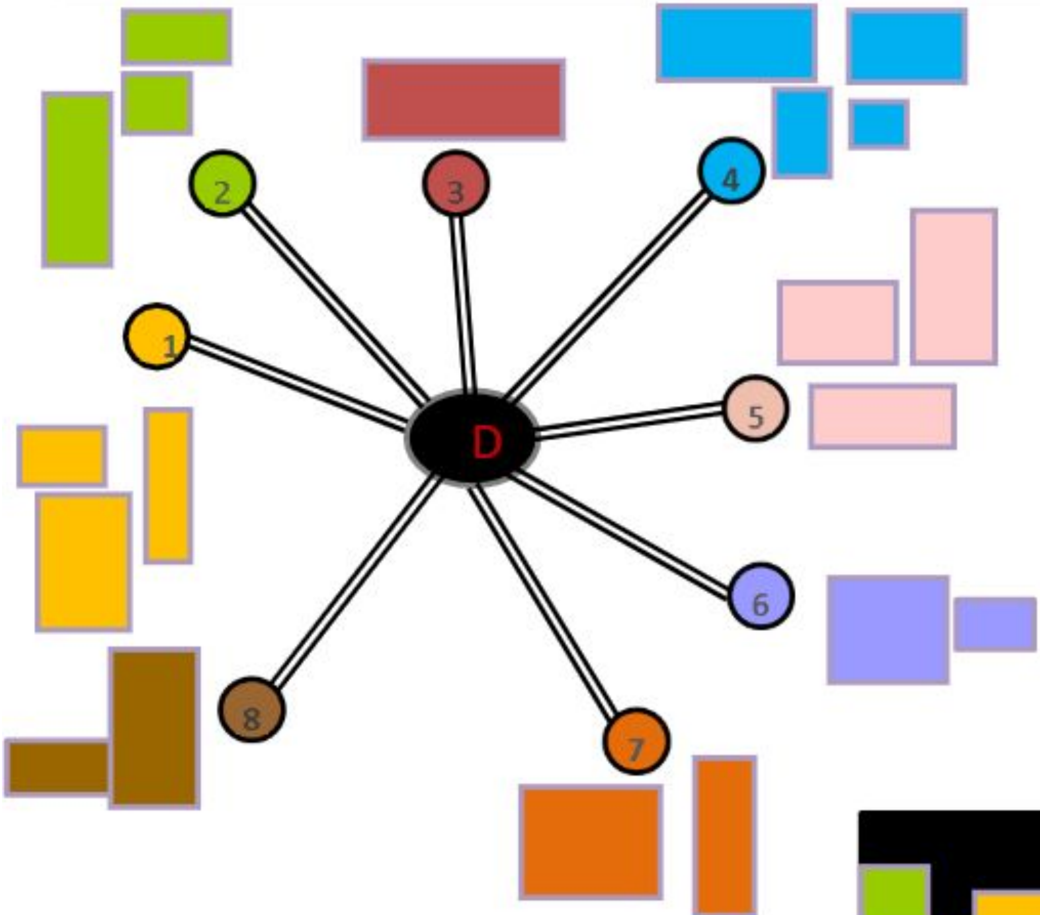
Juan, A. A., Faulin, J., Ruiz, R., Barrios, B., & Caballé, S. (2010). The SR-GCWS hybrid algorithm for solving the capacitated vehicle routing problem. Applied Soft Computing, 10(1), 215-224.





https://www.dropbox.com/sh/oo1yh739yxl9i7n/AACIAea5yjEWTQ97vBdFndBXa?dl=0

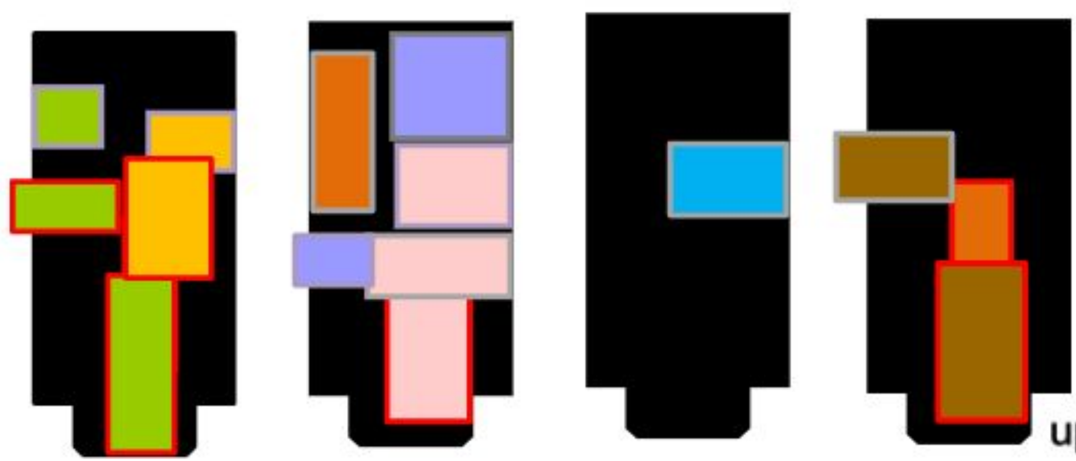
Starting with a Dummy (but feasible) Solution



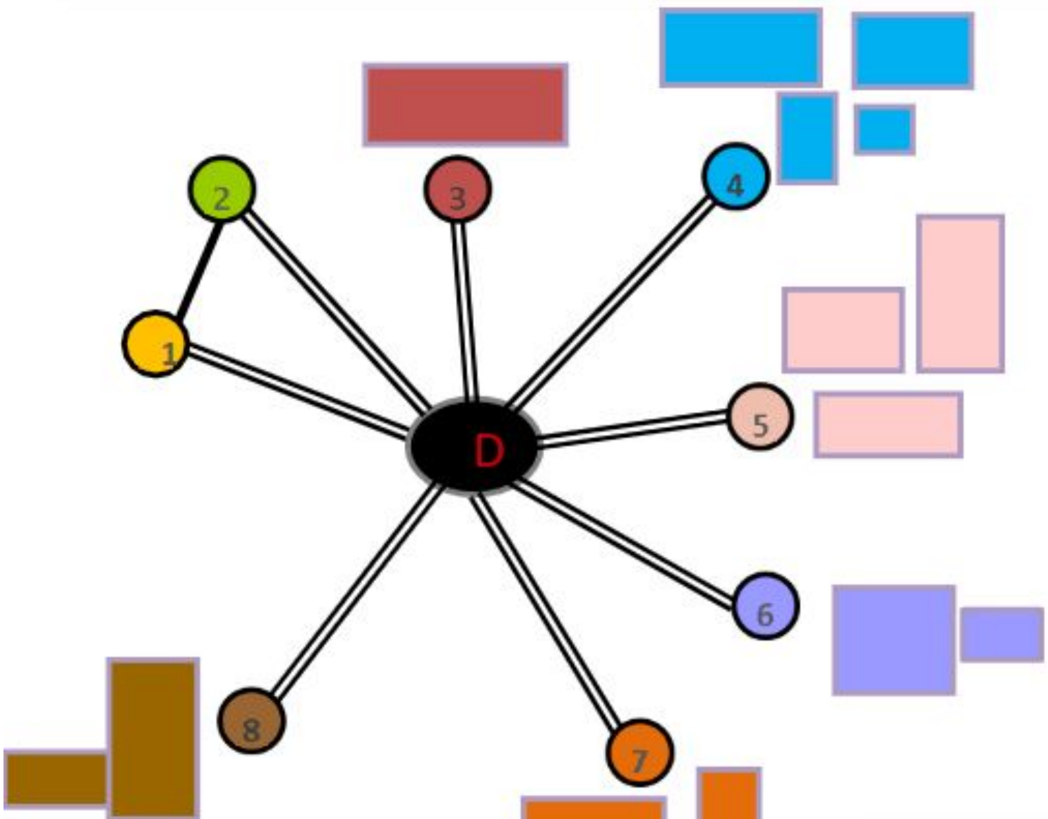
Edge	Saving
(1,2)	
(5,6)	
(6,7)	
(4,5)	
(3,4)	
…	
(1,5)	
(4,8)	

savings of edge (i, j)

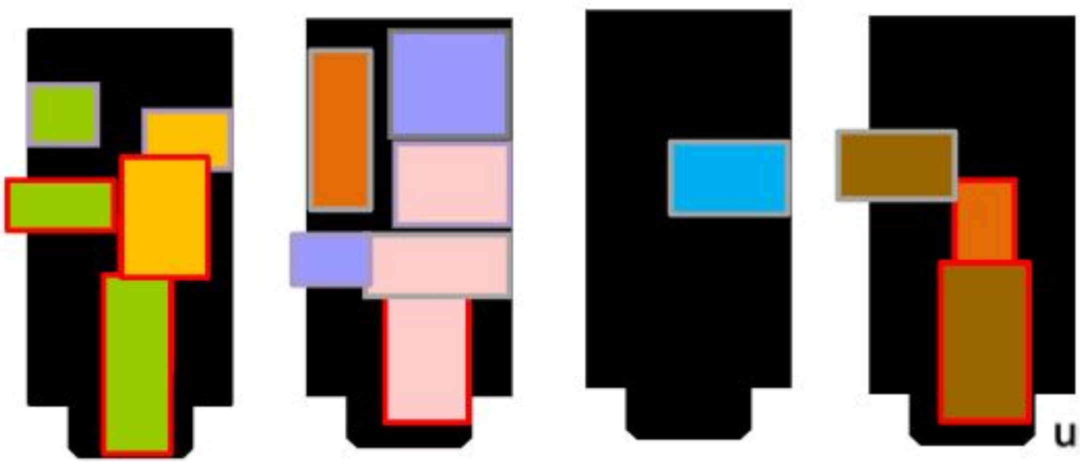
$s(i, j) = c(0, i) + c(0, j) - c(i, j)$



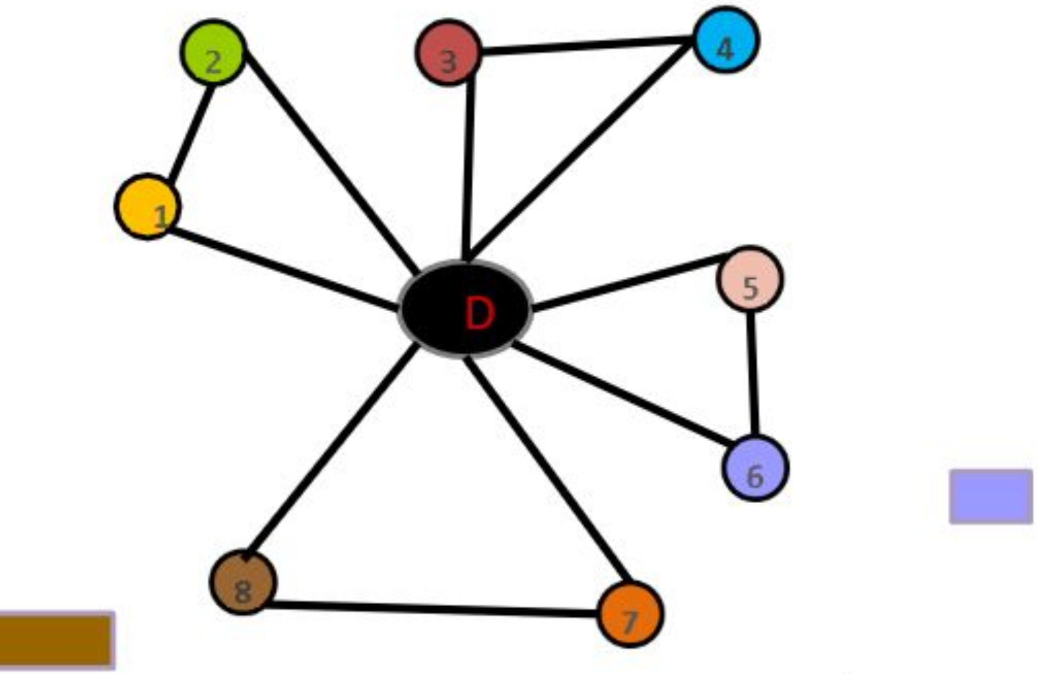
The Next Edge is Selected from the Savings List



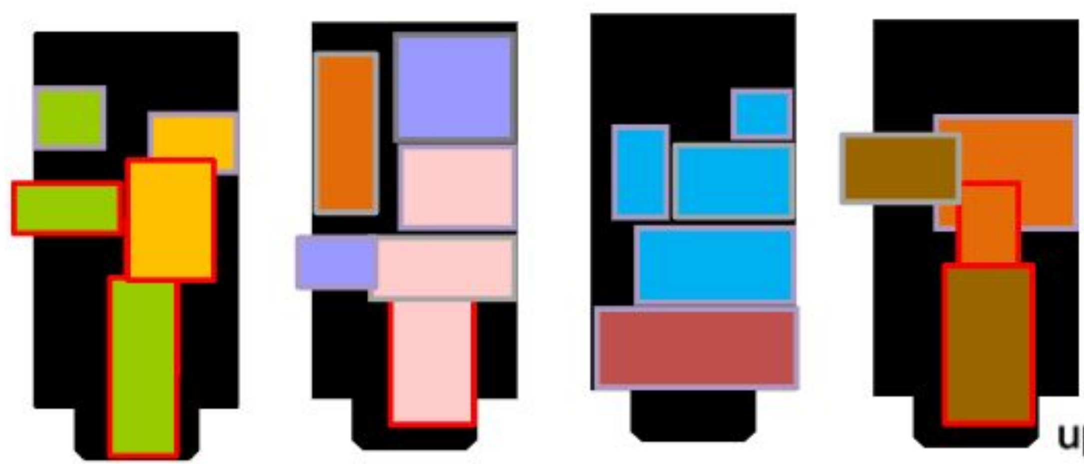
Edge	Saving
(1,2)	
(5,6)	
(6,7)	
(4,5)	
(3,4)	
...	
(1,5)	
(4,8)	



Until the Route-Merging Process is Finished

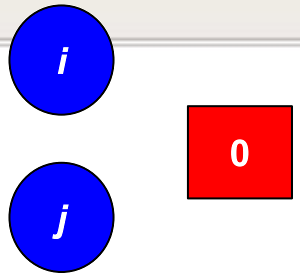


Edge	Saving
(1,2)	
(5,6)	
(6,7)	
(4,5)	
(3,4)	
...	
(1,5)	
(4,8)	

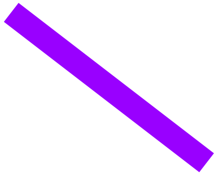


vrp_objects.py Classes Node and Edge

vrp_objects.py cws_heuristic.py



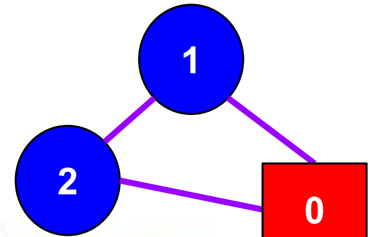

```python
class Node:

    def __init__(self, ID, x, y, demand):
        self.ID = ID # node identifier (depot ID = 0)
        self.x = x # Euclidean x-coordinate
        self.y = y # Euclidean y-coordinate
        self.demand = demand # demand (is 0 for depot and positive for others)
        self.inRoute = None # route to which node belongs
        self.isInterior = False # an interior node is not connected to depot
        self.dnEdge = None # edge (arc) from depot to this node
        self.ndEdge = None # edge (arc) from this node to depot


class Edge:

    def __init__(self, origin, end):
        self.origin = origin # origin node of the edge (arc)
        self.end = end # end node of the edge (arc)
        self.cost = 0.0 # edge cost
        self.savings = 0.0 # edge savings (Clarke & Wright)
        self.invEdge = None # inverse edge (arc)
```

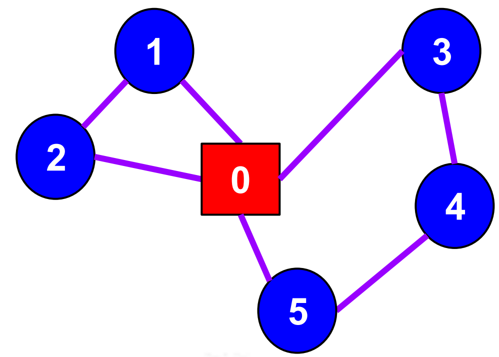


spyder

vrp_objects.py   Classes Route and Solution

```python

class Route:

    def __init__(self):
         self.cost = 0.0 # cost of this route
         self.edges = [] # sorted edges in this route
         self.demand = 0.0 # total demand covered by this route

    def reverse(self): # e.g. 0 -> 2 -> 6 -> 0 becomes 0 -> 6 -> 2 -> 0
        size = len(self.edges)
        for i in range(size):
            edge = self.edges[i]
            invEdge = edge.invEdge
            self.edges.remove(edge)
            self.edges.insert(0, invEdge)


class Solution:

    last_ID = -1 # counts the number of solutions, starts with 0

    def __init__(self):
        Solution.last_ID += 1
        self.ID = Solution.last_ID
        self.routes = [] # routes in this solution
        self.cost = 0.0 # cost of this solution
        self.demand = 0.0 # total demand covered by this solution
```





cws_heuristic.py Reading an Instance Data

vrp_objects.py | cws_heuristic.py

```python
""" Clarke & Wright savings heuristic for the VRP """

from vrp_objects import Node, Edge, Route, Solution
import math
import operator


""" Read instance data from txt file """

vehCap = 100.0 # update vehicle capacity for each instance
instanceName = 'A-n80-k10' # name of the instance
# txt file with the VRP instance data (nodeID, x, y, demand)
fileName = 'data/' + instanceName + '_input_nodes.txt'


with open(fileName) as instance:
    i = 0
    nodes = []
    for line in instance:
        # array data with node data: x, y, demand
        data = [float(x) for x in line.split()]
        aNode = Node(i, data[0], data[1], data[2])
        nodes.append(aNode)
        i += 1

```

Make sure you use the right **`vehCap`** when you test a different instance.

Data on instances is available at: https://www.dropbox.com/sh/uwbixk6iuvxdozg/AADngjZHg765Qd0IDj1kRGSaa?dl=0

JORS Table with Instances and CWS Solutions

Table 1 Results for 33 standard benchmarks (multi-thread C implementation)

Instance	*# nodes*	*vCap*	*CWS Sol.*	*Gap BSK-CWS (%)*	*Best-known Sol.*	*BKS (int)*	*# routes BKS*	*Source*	*Gap BKS-OBS (%)*	*Our Best Sol.*	*# routes OBS*	*Time (s)*
A-n32-k5	32	100	843.69	7.09	[illegible]	[illegible]	[illegible]	[illegible]	−0.09	787.08	5	1
A-n38-k5	38	100	768.14	4.63	[illegible]	[illegible]	[illegible]	[illegible]	−0.03	733.95	5	1
A-n45-k7	45	100	1 199.98	4.59	[illegible]	[illegible]	[illegible]	[illegible]	−0.04	1 146.77	7	1
A-n55-k9	55	100	1 099.84	2.36	[illegible]	[illegible]	[illegible]	[illegible]	0.00	1 074.46	9	1
A-n60-k9	60	100	1 421.88	4.87	[illegible]	[illegible]	[illegible]	[illegible]	0.00	1 355.80	9	1
A-n61-k9	61	100	1 102.23	6.08	[illegible]	[illegible]	[illegible]	[illegible]	0.00	1 039.08	9	2
A-n65-k9	65	100	1 239.42	4.89	1 181.69	1 174	9	2	0.00	1 181.69	9	2
A-n80-k10	80	100	1 860.94	5.35	1 766.50	1 763	10	1	0.00	1 766.50	10	178
B-n50-k7	50	100	748.80	0.54	744.78	741	7	1	−0.07	744.23	7	1
B-n52-k7	52	100	764.90	1.98	750.08	747	7	1	−0.02	749.96	7	1
B-n57-k9	57	100	1 653.42	3.10	1 603.63	1 598	9	1	−0.08	1 602.28	9	1
B-n78-k10	78	100	1 264.56	2.87	1 229.27	1 221	10	1	−0.11	1 227.90	10	9
E-n22-k4	22	6 000	388.77	3.59	375.28	375	4	2	0.00	375.28	4	1
E-n30-k3	30	4 500	534.45	−0.25	535.80	534	3	2	−5.75	505.01	4	1
E-n33-k4	33	8 000	843.10	0.52	838.72	835	4	2	−0.13	837.67	4	1
E-n51-k5	51	160	584.64	11.37	524.94	521	5	1	−0.06	524.61	5	1
E-n76-k7	76	220	737.74	7.29	687.60	682	7	2	0.00	687.60	7	7
E-n76-k10	76	140	900.26	7.51	837.36	830	10	1	−0.25	835.28	10	231
E-n76-k14	76	100	1 073.43	4.55	1 026.71	1 021	14	1	−0.22	1 024.40	15	32
F-n45-k4	45	2 010	739.02	1.99	724.57	721	4	3	−0.14	723.54	4	5
F-n72-k4	72	30 000	256.19	2.97	248.81	237	4	3	−2.75	241.97	4	2
F-n135-k7	135	2 210	1 219.32	4.16	1 170.65	1 159	7	1	−0.51	1 164.73	7	131
M-n101-k10	101	200	833.51	1.67	819.81	820	10	3	−0.03	819.56	10	0
M-n121-k7	121	200	1 068.14	2.20	1 045.16	1 034	7	1	−0.12	1 043.88	7	107
P-n22-k8	22	3 000	590.62	−1.80	601.42	603	8	3	−2.10	588.79	9	1
P-n40-k5	40	140	518.37	12.27	461.73	458	5	3	0.00	461.73	5	1
P-n50-k10	50	100	734.32	4.97	699.56	696	10	3	0.00	699.56	10	2
P-n55-k15	55	70	978.07	−1.35	991.48	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
P-n65-k10	65	130	851.67	6.90	796.67	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
P-n70-k10	70	135	896.86	8.05	830.02	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
P-n76-k4	76	350	689.13	15.20	598.22	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
P-n76-k5	76	280	698.51	9.99	635.04	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
P-n101-k4	101	400	765.38	10.56	692.28	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Averages	63			4.87					[illegible]			[illegible]

The A-n80-k10 uses a vehCap = 100, and its CWS cost is 1860.94.

Juan, A. A., Faulín, J., Jorba, J., Riera, D., Masip, D., & Barrios, B. (2011). On the use of Monte Carlo simulation, cache and splitting techniques to improve the Clarke and Wright savings heuristics. Journal of the Operational Research Society, 62(6), 1085-1097.

cws_heuristic.py   Creating the Edges and List

```python
""" Construct edges with costs and savings list from nodes """

depot = nodes[0] # node 0 is the depot

for node in nodes[1:]: # excludes the depot
    dnEdge = Edge(depot, node) # creates the (depot, node) edge (arc)
    ndEdge = Edge(node, depot)
    dnEdge.invEdge = ndEdge # sets the inverse edge (arc)
    ndEdge.invEdge = dnEdge
    # compute the Euclidean distance as cost
    dnEdge.cost = math.sqrt((node.x - depot.x)**2 + (node.y - depot.y)**2)
    ndEdge.cost = dnEdge.cost # assume symmetric costs
    # save in node a reference to the (depot, node) edge (arc)
    node.dnEdge = dnEdge
    node.ndEdge = ndEdge

savingsList = []
for i in range(1, len(nodes) - 1): # excludes the depot
    iNode = nodes[i]
    for j in range(i + 1, len(nodes)):
        jNode = nodes[j]
        ijEdge = Edge(iNode, jNode) # creates the (i, j) edge
        jiEdge = Edge(jNode, iNode)
        ijEdge.invEdge = jiEdge # sets the inverse edge (arc)
        jiEdge.invEdge = ijEdge
        # compute the Euclidean distance as cost
        ijEdge.cost = math.sqrt((jNode.x - iNode.x)**2 + (jNode.y - iNode.y)**2)
        jiEdge.cost = ijEdge.cost # assume symmetric costs
        # compute savings as proposed by Clark & Wright
        ijEdge.savings = iNode.ndEdge.cost + jNode.dnEdge.cost - ijEdge.cost
        jiEdge.savings = ijEdge.savings
        # save one edge in the savings list
        savingsList.append(ijEdge)
# sort the list of edges from higher to lower savings
savingsList.sort(key = operator.attrgetter("savings"), reverse = True)
```

cws_heuristic.py   Dummy Sol and Aux. Funct.

```python
""" Construct the dummy solution """

sol = Solution()
for node in nodes[1:]: # excludes the depot
    dnEdge = node.dnEdge # get the (depot, node) edge
    ndEdge = node.ndEdge
    dndRoute = Route() # construct the route (depot, node, depot)
    dndRoute.edges.append(dnEdge)
    dndRoute.demand += node.demand
    dndRoute.cost += dnEdge.cost
    dndRoute.edges.append(ndEdge)
    dndRoute.cost += ndEdge.cost
    node.inRoute = dndRoute # save in node a reference to its current route
    node.isInterior = False # this node is currently exterior (connected to depot)
    sol.routes.append(dndRoute) # add this route to the solution
    sol.cost += dndRoute.cost
    sol.demand += dndRoute.demand


""" Perform the edge-selection & routing-merging iterative process """

def checkMergingConditions(iNode, jNode, iRoute, jRoute):
    # condition 1: iRoute and jRoute are not the same route object
    if iRoute == jRoute: return False
    # condition 2: both nodes are exterior nodes in their respective routes
    if iNode.isInterior == True or jNode.isInterior == True: return False
    # condition 3: demand after merging can be covered by a single vehicle
    if vehCap < iRoute.demand + jRoute.demand: return False
    # else, merging is feasible
    return True


def getDepotEdge(aRoute, aNode):
    ''' returns the edge in aRoute that contains aNode and the depot
      (it will be the first or the last one) '''
    # check if first edge in aRoute contains aNode and depot
    origin = aRoute.edges[0].origin
    end = aRoute.edges[0].end
    if ((origin == aNode and end == depot) or
        (origin == depot and end == aNode)):
            return aRoute.edges[0]
    else: # return last edge in aRoute
        return aRoute.edges[-1]
```

spyder

cws_heuristic.py Iterative Merging Process

```python
while len(savingsList) > 0: # list is not empty
    ijEdge = savingsList.pop(0) # select the next edge from the list
    # determine the nodes i < j that define the edge
    iNode = ijEdge.origin
    jNode = ijEdge.end
    # determine the routes associated with each node
    iRoute = iNode.inRoute
    jRoute = jNode.inRoute
    # check if merge is possible
    isMergeFeasible = checkMergingConditions(iNode, jNode, iRoute, jRoute)
    # if all necessary conditions are satisfied, merge
    if isMergeFeasible == True:
        # iRoute will contain either edge (depot, i) or edge (i, depot)
        iEdge = getDepotEdge(iRoute, iNode) # iEdge is either (0,i) or (i,0)
        # remove iEdge from iRoute and update iRoute cost
        iRoute.edges.remove(iEdge)
        iRoute.cost -= iEdge.cost
        # if there are multiple edges in iRoute, then i will be interior
        if len(iRoute.edges) > 1: iNode.isInterior = True
        # if new iRoute does not start at 0 it must be reversed
        if iRoute.edges[0].origin != depot: iRoute.reverse()
        # jRoute will contain either edge (depot, j) or edge (j, depot)
        jEdge = getDepotEdge(jRoute, jNode) # jEdge is either (0,j) or (j,0)
        # remove jEdge from jRoute and update jRoute cost
        jRoute.edges.remove(jEdge)
        jRoute.cost -= jEdge.cost
        # if there are multiple edges in jRute, then j will be interior
        if len(jRoute.edges) > 1: jNode.isInterior = True
        # if new jRoute starts at 0 it must be reversed
        if jRoute.edges[0].origin == depot: jRoute.reverse()
        # add ijEdge to iRoute
        iRoute.edges.append(ijEdge)
        iRoute.cost += ijEdge.cost
        iRoute.demand += jNode.demand
        jNode.inRoute = iRoute
        # add jRoute to new iRoute
        for edge in jRoute.edges:
            iRoute.edges.append(edge)
            iRoute.cost += edge.cost
            iRoute.demand += edge.end.demand
            edge.end.inRoute = iRoute
        # delete jRoute from emerging solution
        sol.cost -= ijEdge.savings
        sol.routes.remove(jRoute)
```

cws_heuristic.py Printing the CWS Solution

```python
print('Cost of C&W savings sol =', "{:.{}f}".format(sol.cost, 2))
for route in sol.routes:
    s = str(0)
    for edge in route.edges:
        s = s + '-' + str(edge.end.ID)
    print('Route: ' + s + ' || cost = ' + "{:.{}f}".format(route.cost, 2))
```

Console 1/A

```
Python 3.7.6 (default, Jan  8 2020, 19:59:22)
Type "copyright", "credits" or "license" for more information.

IPython 7.12.0 -- An enhanced Interactive Python.

In [1]: runfile('/home/aajp/2020_VRP_CWS_heuristic_Python/cws_heuristic.py', wdir='/home/aajp/
2020_VRP_CWS_heuristic_Python')
Cost of C&W savings sol = 1860.94
Route: 0-1-63-10-7-21-0 || cost = 130.70
Route: 0-64-17-31-27-59-5-23-62-0 || cost = 188.89
Route: 0-24-6-30-78-8-37-2-34-71-0 || cost = 220.95
Route: 0-11-52-28-79-18-48-14-0 || cost = 193.22
Route: 0-13-74-39-60-29-44-12-0 || cost = 143.38
Route: 0-46-20-75-25-41-15-55-9-54-72-0 || cost = 248.96
Route: 0-33-47-56-69-65-35-26-19-57-61-16-43-68-0 || cost = 307.36
Route: 0-36-38-66-67-53-3-77-51-0 || cost = 145.84
Route: 0-40-42-73-49-0 || cost = 87.05
Route: 0-70-76-50-45-22-4-32-58-0 || cost = 194.59
```

Plotting the CWS Solution with Networkx (1/4)

Console 1/A

```
In [1]: runfile('/home/aajp/2020_VRP_CWS_heuristic_Python/cws_heuristic.py', wdir='/home/aajp/
2020_VRP_CWS_heuristic_Python')
Instance:   A-n32-k5
Cost of C&W savings sol = 843.69
Route: 0-12-1-13-7-16-0 || cost = 111.92
Route: 0-23-2-3-17-19-31-21-0 || cost = 213.22
Route: 0-14-22-9-8-11-4-28-18-6-26-0 || cost = 257.90
Route: 0-24-30-0 || cost = 65.60
Route: 0-27-29-15-10-25-5-20-0 || cost = 195.05
```

This is the A-n32-k5 with just 32 nodes.

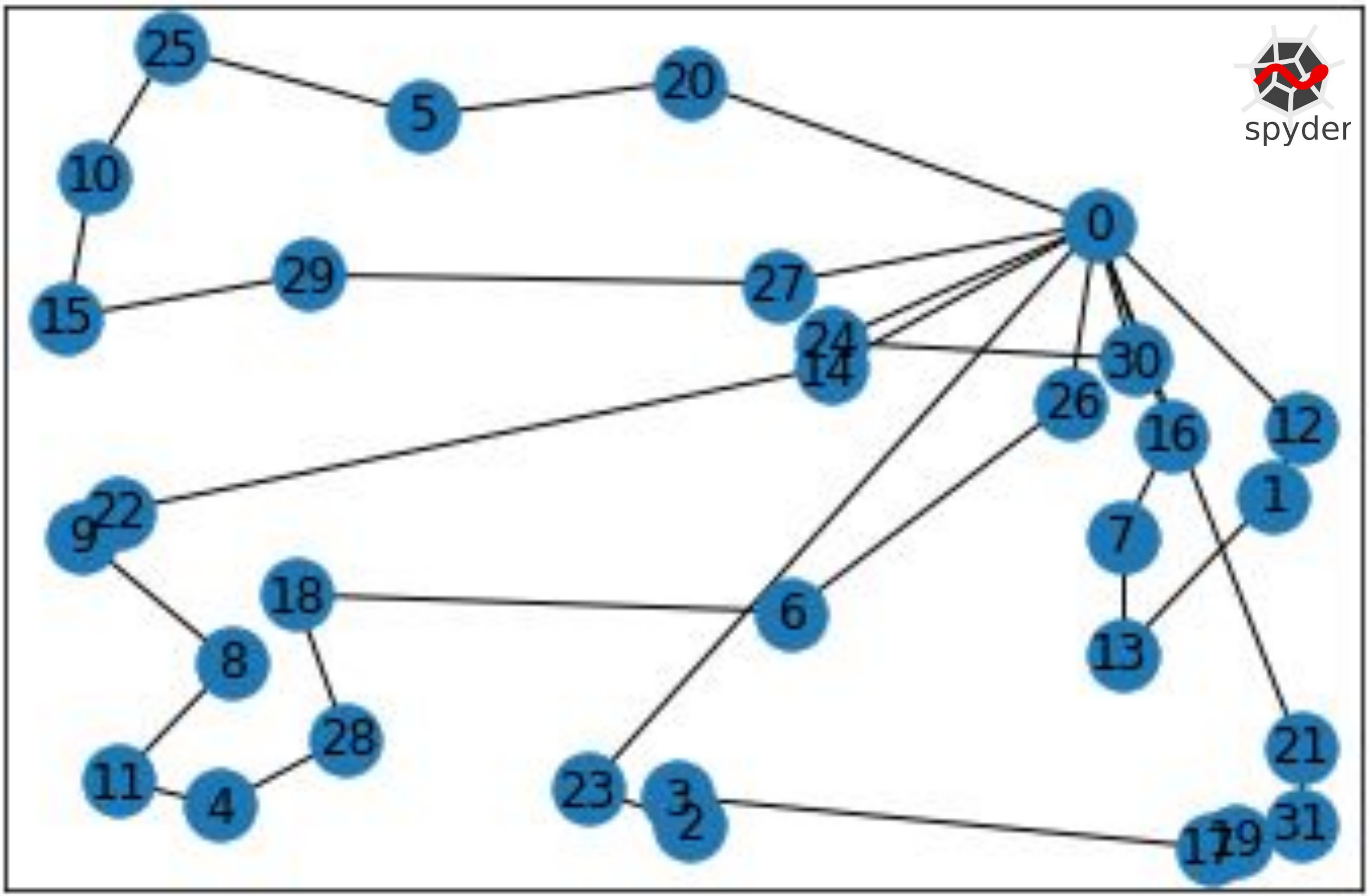
Import `networkx` at the beginning of your code.

```python
import networkx as nx
```

```python
G = nx.Graph()
for route in sol.routes:
    for edge in route.edges:
        G.add_edge(edge.origin.ID, edge.end.ID)
        G.add_node(edge.end.ID, coord=(edge.end.x, edge.end.y))

coord = nx.get_node_attributes(G, 'coord')
nx.draw_networkx(G, coord)
```

spyder

Plotting the CWS Solution with Networkx (2/4)



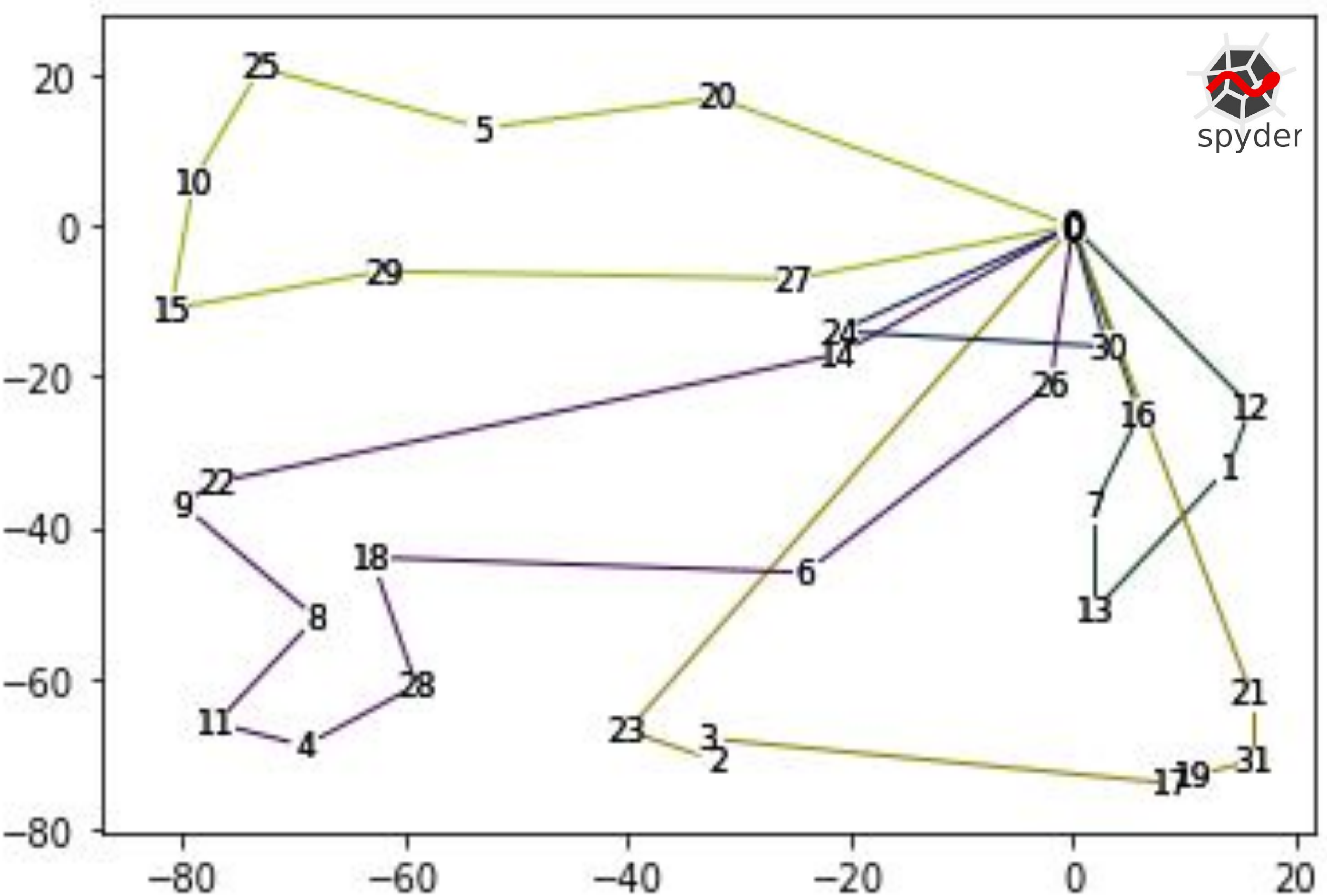
Plotting the CWS Solution with Networkx (3/4)

```python
import networkx as nx
import matplotlib.pyplot as plt
import random
```

Optional (if you want a nicer plot…)

```python
G = nx.Graph()
fnode = sol.routes[0].edges[0].origin
G.add_node(fnode.ID, coord=(fnode.x,fnode.y))
coord = nx.get_node_attributes(G,'coord')
fig, ax = plt.subplots() #Add axes
nx.draw_networkx_nodes(G, coord, node_size = 60, node_color='white', ax = ax)
nx.draw_networkx_labels(G, coord)

j=0
for route in sol.routes:
    #Assign random colors in RGB
    c1 = int(random.uniform(0, 255)) if (j%3 == 2) else (j%3)*int(random.uniform(0, 255))
    c2 = int(random.uniform(0, 255)) if ((j+1)%3 == 2) else ((j+1)%3)*int(random.uniform(0, 255))
    c3 = int(random.uniform(0, 255)) if ((j+2)%3 == 2) else ((j+2)%3)*int(random.uniform(0, 255))
    for edge in route.edges:
        G.add_edge(edge.origin.ID, edge.end.ID)
        G.add_node(edge.end.ID, coord=(edge.end.x, edge.end.y))
        coord = nx.get_node_attributes(G,'coord')
        nx.draw_networkx_nodes(G, coord, node_size = 60, node_color='white', ax = ax)
        nx.draw_networkx_edges(G, coord, edge_color='#%02x%02x%02x' % (c1, c2, c3))
        nx.draw_networkx_labels(G, coord, font_size = 9)
        G.remove_node(edge.origin.ID)
    j += 1

limits=plt.axis('on') #Turn on axes
ax.tick_params(left=True, bottom=True, labelleft=True, labelbottom=True)
```

Plotting the CWS Solution with Networkx (4/4)



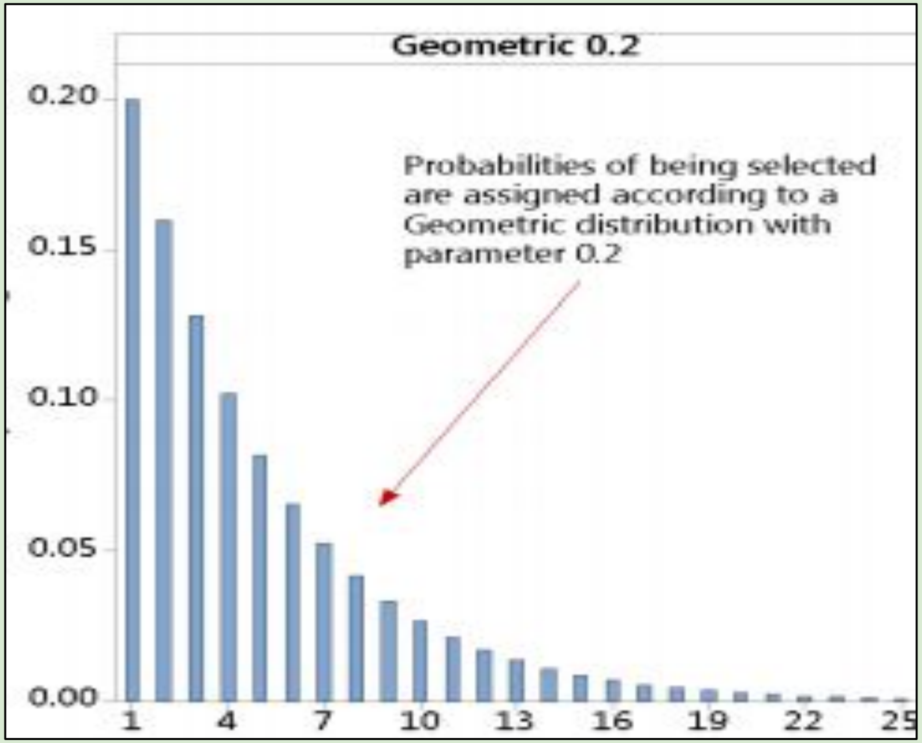
Homework Activities

1. Construct your own **Python program** to implement the CWS heuristic for solving the VRP and test it in different instances, comparing your results with the ones for the CWS solution provided in the JORS table.

2. (Optional) Construct a multi-start biased-randomized version of the CWS heuristic.

3. (Optional) Complete a comparative study to illustrate how the biased-randomized algorithm (BRA) outperforms the CWS heuristic.

4. (Optional) Add a dictionary (hash map) to remember the best-found way to route a given set of nodes. This might improve the BRA performance.



References

Barry, P. (2016). Head First Python: A Brain-Friendly Guide. O'Reilly Media, Inc.

Brownlee, J. (2011). Clever algorithms: nature-inspired programming recipes. Jason Brownlee.

Downey, A. B. (2015): Think Python: How to Think Like a Computer Scientist. O'Reilly Media

Johnson, M. J. (2018). A concise introduction to programming in Python. CRC Press.

Labadie, N., Prins, C., Prodhon, C., Monmarché, N., & Siarry, P. (2016). Metaheuristics for vehicle routing problems. ISTE Limited.

Panyam, S. (2011). Clever Algorithms in Python.

Toth, P., & Vigo, D. (Eds.). (2014). Vehicle routing: problems, methods, and applications. Society for Industrial and Applied Mathematics.