Generate Map Data

July 15, 2024

```
[1]: import matplotlib.pyplot as plt
     import math
     import random
     import numpy as np
```

Hints for students: The utility section contains code you need to use *unchanged* to generate the test data required. You may use this code for your final solution, beware not to overwrite function definitions in this section. Otherwise you can ignore the code in this section.

1 Utilities

Note: Section 1.1-1.4 is identical to the MST example (Week 3). 1.5-1.10 are adjustments of the data structures for the current problem.

1.1 Points and Distances

Euclidean Distance between two points

```
[2]: def dist(p1, p2):
         (x1, y1) = p1
         (x2, y2) = p2
         return int(math.sqrt((x1-x2)**2+(y1-y2)**2))
```

The nearest link between two point sets

```
[3]: def nearest(X, P):
         minD = math.inf
         minP = None
         for p in P:
             for x in X:
                 d=dist(x, p)
                 if d<minD:
                     minX, minP, minD = x, p, d
         return minX, minP
```

1.2 Graphs

```
[4]: def generateRandomGraph(n, r):

         def rounding(x):
             return int(math.floor(x/10))*10

         r = rounding(r)
         x0 = r
         y0 = r
         gridsize = rounding(r / math.sqrt(n) * 1.4)
         r = r//gridsize*gridsize
         split = 2*r//gridsize+1
         X = np.linspace(x0-r, x0+r, split)
         Y = np.linspace(y0-r, y0+r, split)
         P = [ (int(x), int(y)) for x in X for y in Y if dist((x,y), (x0,y0)) <= r*1.
     ↪2 ]
         P = random.sample(P, k=n)

         E = []

         def addEdge(p, q):
             if p in P and q in P and (p, q) not in E and (q, p) not in E:
                 E.append((p, q))
         def addDiagonalEdge(p, q):
             (xp, yp) = p
             (xq, yq) = q
             if p in P and q in P and (xp, yq) not in P and (xq, yp) not in P and␣
     ↪(p, q) not in E and (q, p) not in E:
                 E.append((p, q))

         for (x, y) in P:
             addEdge( (x, y), (x, y+gridsize) )
             addEdge( (x, y), (x, y-gridsize) )
             addEdge( (x, y), (x+gridsize, y) )
             addEdge( (x, y), (x-gridsize, y) )
             addDiagonalEdge( (x, y), (x+gridsize, y+gridsize) )
             addDiagonalEdge( (x, y), (x+gridsize, y-gridsize) )
             addDiagonalEdge( (x, y), (x-gridsize, y+gridsize) )
             addDiagonalEdge( (x, y), (x-gridsize, y-gridsize) )

         return sorted(P), sorted(E)
```

```
[5]: def plotGraph(P, E, col='b', grid=False):
         fig = plt.gcf()
         fig.set_size_inches(6, 6)
         if not grid:
```
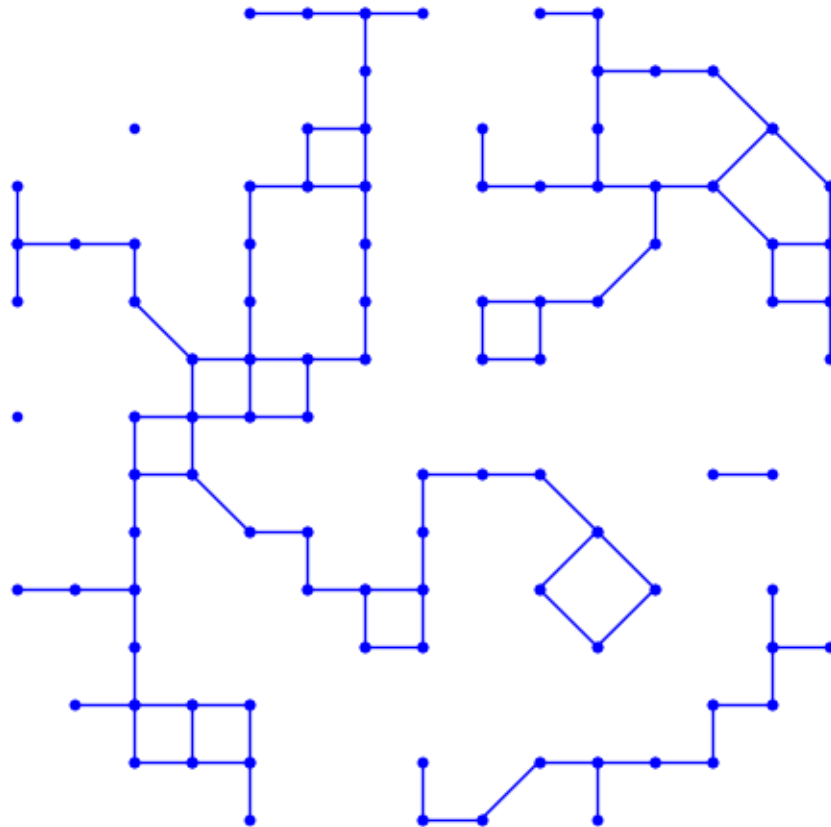
```
        plt.axis('off')
    plt.plot( [ p[0] for p in P ], [ p[1] for p in P ], col+'o', lw=1, ms=3)
    for (p, q) in E:
        plt.plot( [ p[0], q[0] ], [ p[1], q[1] ], col+'-o', lw=1, ms=3)
    if grid:
        plt.grid()
```

[6]:
```
random.seed(42)
V, E = generateRandomGraph(100, 4500)
plotGraph(V, E)
```


The random generation may result in a graph consisting of multiple not connected subgraphs. Split a graph into not connected subgraphs, if any.

[7]:
```
def subgraph(P, E):
    P = P.copy()
    E = E.copy()
    PP = [ P[0] ]
    EE = []
```

```
            P = P[1:]
        extended = True
        while extended:
            extended = False
            for (a, b) in E:
                if a in PP and b in P:
                    PP.append(b)
                    P.remove(b)
                    EE.append((a, b))
                    E.remove((a, b))
                    extended = True
                    break
                if a in P and b in PP:
                    PP.append(a)
                    P.remove(a)
                    EE.append((a, b))
                    E.remove((a, b))
                    extended = True
                    break
                if a in PP and b in PP:
                    EE.append((a, b))
                    E.remove((a, b))
                    extended = True
                    break
        return PP, EE, P, E
```
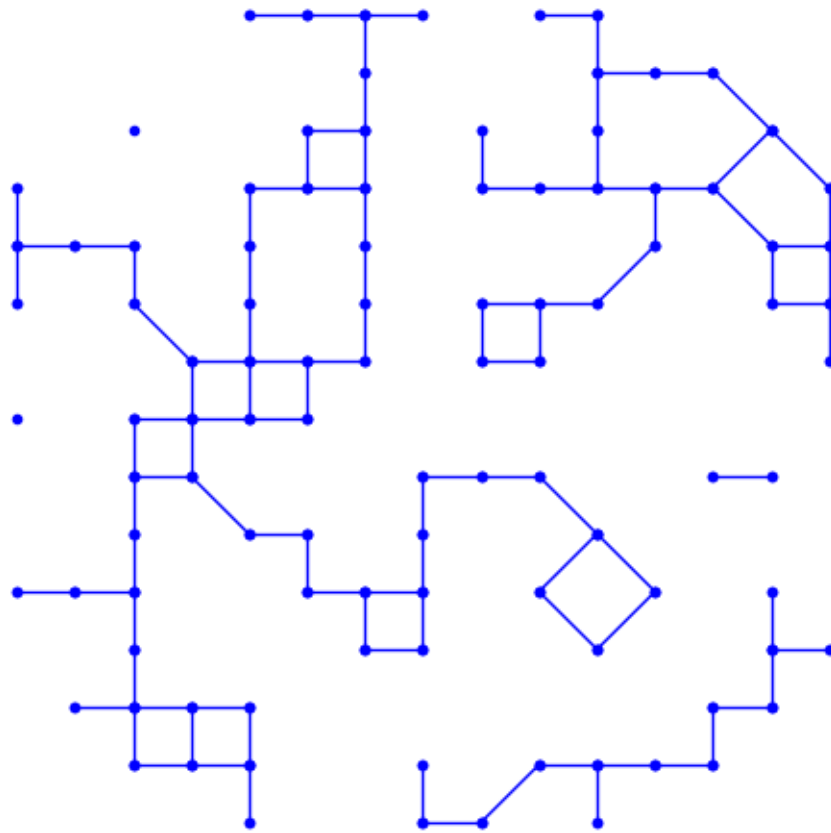
[8]:
```
def generateGraph(n, r):
    P, E = generateRandomGraph(n, r)
    P0, _, P1, _ = subgraph(P, E)
    while len(P1)>0:
        (p, q) = nearest(P0, P1)
        E.append((p, q))
        P0, _, P1, _ = subgraph(P, E)
    return P, E
```
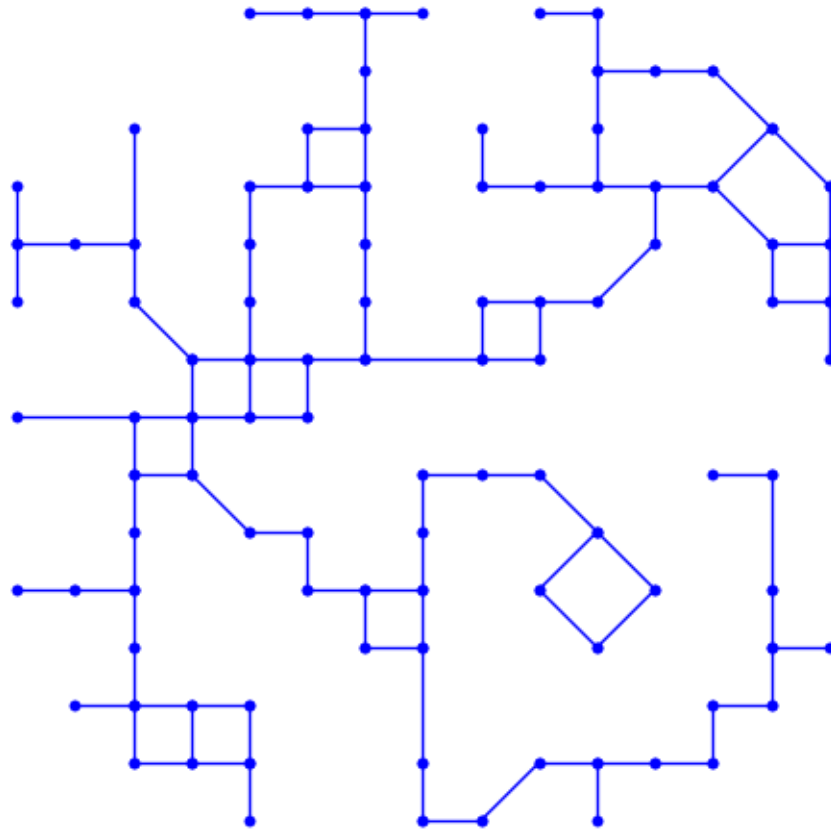
[9]:
```
random.seed(42)
V, E = generateRandomGraph(100, 4500)
plotGraph(V, E)
```
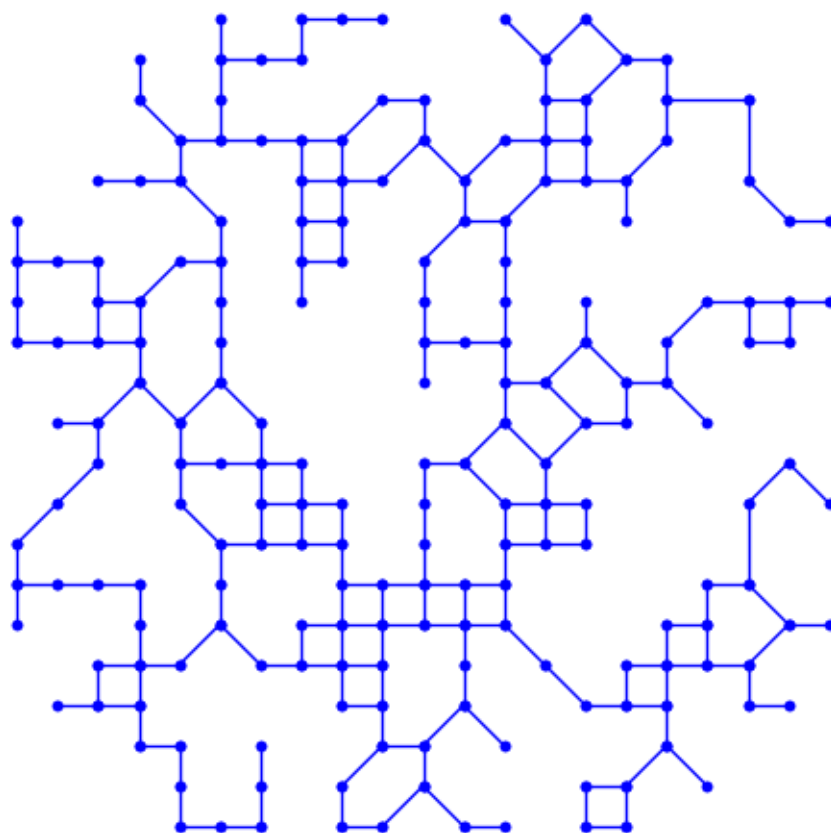
```
[10]: random.seed(42)
      V, E = generateGraph(100, 4500)
      plotGraph(V, E)
```
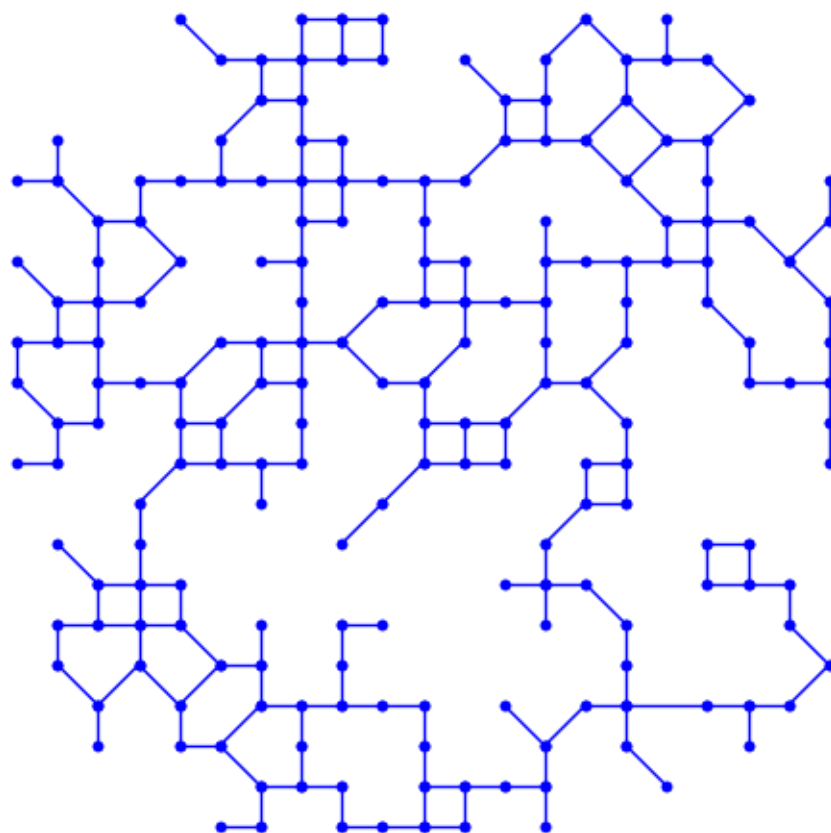
```
[11]: for i in range(7000, 7005):
          random.seed(i)
          P, E = generateGraph(200, 4000)
          print("Graph for seed", i, "has", len(P), "vertices and", len(E), "edges")
          plotGraph(P, E)
          plt.show()
```
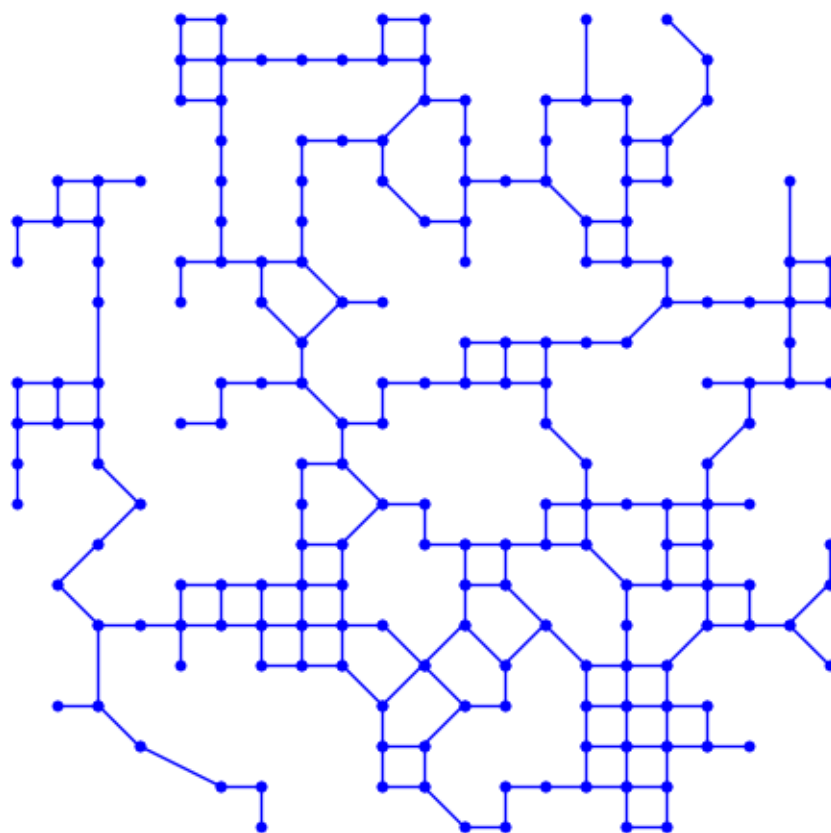
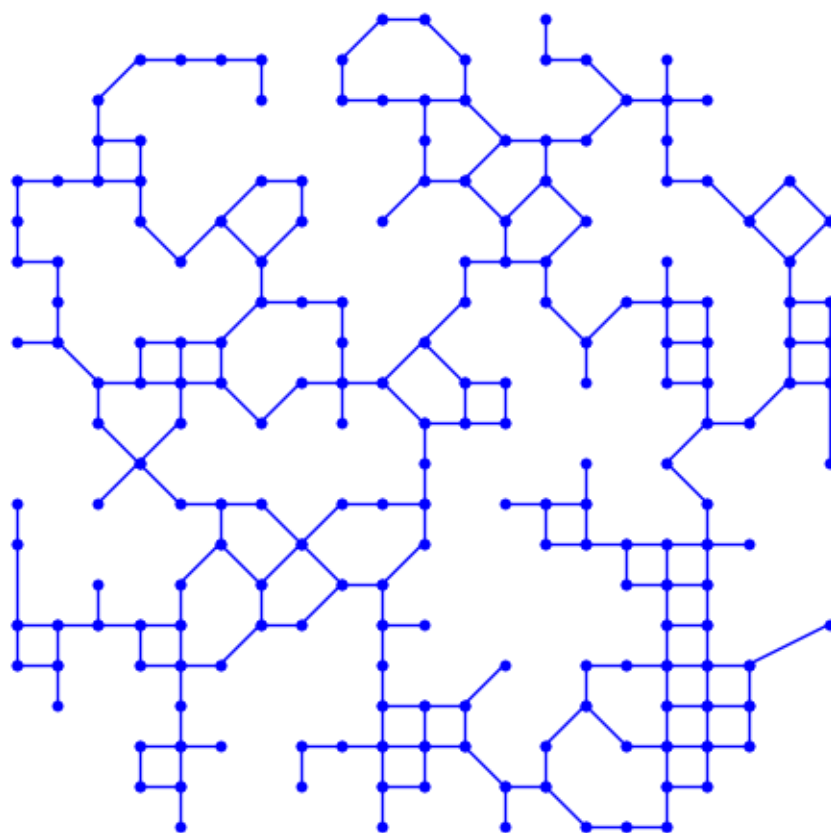Graph for seed 7000 has 200 vertices and 241 edges

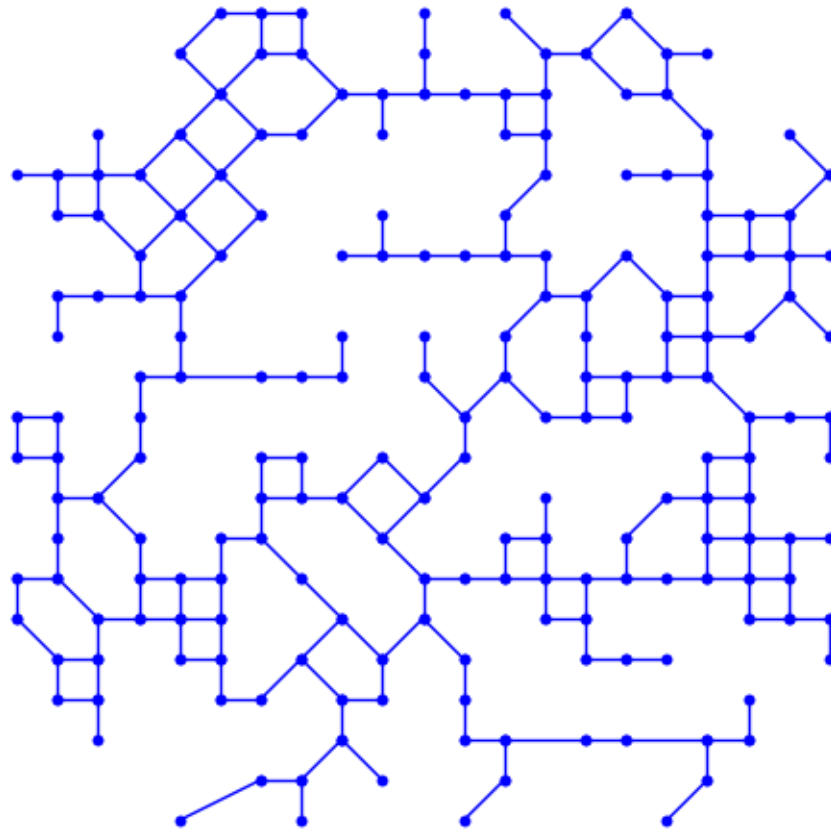Graph for seed 7001 has 200 vertices and 239 edges

Graph for seed 7002 has 200 vertices and 248 edges

Graph for seed 7003 has 200 vertices and 242 edges

Graph for seed 7004 has 200 vertices and 241 edges

1.3 Lists and Paths

```
[12]: def pathLength(P):
          return 0 if len(P)<=1 else \
                  dist(P[0], P[1])+pathLength(P[1:])
```

```
[13]: def reverse(P):
          return [ P[-i] for i in range(1,len(P)+1) ]
```

```
[14]: def index(x, L):
          for i in range(len(L)):
              if x==L[i]:
                  return i
          return None
```

```
[15]: def addWithoutDuplicates(L, X):
          for i in range(len(X)):
              if X[i] not in L:
```

```
                L.append(X[i])
        return L
```

[16]:
```python
def totalLength(edges):
    return sum([ dist(A, B) for A,B in edges ])
```

1.4 Generate Customer Locations

[17]:
```python
def splitEdgeRandomly(V, E, s):
    A, B = s
    p = random.uniform(0.3,0.7)
    x = int(A[0]+p*(B[0]-A[0]))
    y = int(A[1]+p*(B[1]-A[1]))
    t = (x,y)
    E.remove(s)
    E.append((A, t))
    E.append((t, B))
    V.append(t)
    return (V, E), t
```

[18]:
```python
def generateRandomTargets(M, n=5):
    V, E = M
    V, E = V.copy(), E.copy()
    T = []
    # we want to ensure that the beginning of the
    # sequence of points generated randomly stays
    # the same
    mindist = 200
    while len(T)<n:
        s = random.choice(E)
        A, B = s
        if dist(A,B)>mindist: # avoid targets placed narrowly
            (V, E), t = splitEdgeRandomly(V, E, s)
            T.append(t)
    return sorted(T)
```

[19]:
```python
def addTargets(M, T):
    V, E = M
    E = E.copy()
    V = V.copy()
    for t in T:
        minD = math.inf
        minE = None
        for e in E:
            P, Q = e
            distT = dist(P, t)+dist(t, Q)-dist(P, Q)
            if distT < minD:
```

```
                minD = distT
                minE = e
        P, Q = minE
        E.remove( (P, Q) )
        E.append( (P, t) )
        E.append( (t, Q) )
        V.append(t)
    return V, E
```

1.5 Plot Map with Delivery Route

```
[21]: def plotMap(G, T=[], P=[], w=None,
                style='r-o', lw=1, ms=3,
                styleT='go', msT=3,
                styleP='b-o', lwP=2, msP=3,
                stylePT='go', msPT=7,
                styleW='ro', msW=9,
                text=None, grid=False, labels=False, scale=False):

          V, E = G

          def round_down(x, level): return (x//level)*level
          def round_up(x, level): return (x//level+1)*level

          xmin = round_down(min([ x for (x, _) in V ]), 100)
          xmax = round_up(max([ x for (x, _) in V ]), 100)
          ymin = round_down(min([ y for (_, y) in V ]), 100)
          ymax = round_up(max([ y for (_, y) in V ]), 100)
          dx = xmax-xmin
          dy = ymax-ymin
          yoffset = (ymax-ymin)//10

          fig = plt.gcf()
          fig.set_size_inches(4, 4)
          plt.xlim(xmin, xmax+yoffset)
          plt.ylim(ymin-yoffset, ymax)

          if not grid:
              plt.axis('off')

          for e in E:
              p1, p2 = e
              plt.plot( [ p1[0], p2[0] ],
                        [ p1[1], p2[1] ],
                        style, lw=lw, ms=ms)
          if scale:
              # plot 1000m scale
```

```
        ybar = ymin-0.9*yoffset
        D = [ (xmin, ybar+50), (xmin, ybar), (xmin+1000, ybar), (xmin+1000,␣
␣ybar+50) ]
        plt.plot( [ d[0] for d in D ], [ d[1] for d in D ], 'k-', lw=0.5)
        plt.text(xmin+500, ymin-0.7*yoffset, '1000m' ,␣
␣horizontalalignment='center', size=8)

    if labels:
        for i in range(len(V)):
            x, y = V[i]
            plt.text(x+0.0150*dx, y-0.0350*dy, label(i), size=8)

    for t in T:
        plt.plot( [ t[0] ], [ t[1] ],
                  styleT, ms=msT)

    plt.plot( [ p[0] for p in P ],
              [ p[1] for p in P ],
              styleP, lw=lwP, ms=msP)

    for p in P:
        if p in T:
            plt.plot( [ p[0] ], [ p[1] ],
                      stylePT, ms=msPT)
    if w is not None:
        plt.plot( [ w[0] ], [ w[1] ],
                  styleW, ms=msW)
    if text is not None:
        plt.text(xmax, ymin-0.7*yoffset, text, horizontalalignment='right',␣
␣size=8)
    if grid:
        plt.grid()
    plt.show()
```

1.6 Generate Data

```
[22]: def generateData(seed=None, nodes=35, customers=100,
                       plot=False, log=False):

          if seed is None:

              print("Usage:  M, C = generateData(seed=None, ")
              print("                            nodes=35, customers=100, ")
              print("                            plot=False, log=False)")
              print("")
              print("  seed  the seed value to be used for data generation. ")
              print("        To test the application use seed=0, it will create")
```

14

```python
        print("          a small map, with a very few customer locations.")
        print("")
        print("  nodes the number of intersections (nodes, vertices) in the␣
↪generated map")
        print("")
        print("  customers  the number of customers generated on the map")
        print("")
        print("  log    Controls print output during data generation.")
        print("")
        print("  plot  Controls graphical output during data generation.")
        print("")
        print("Returns:")
        print("")
        print("  M = (V, E) is the generated map given as a graph")
        print("    where V is a list of vertices, with each vertice ")
        print("    given as a pair (x, y) of integer coordinates, ")
        print("    and E is a list of edges, with each edge given")
        print("    as a pair (A, B) of vertices, with each vertex again")
        print("    given as a pair (x, y) of integer coordinates")
        print("")
        print("  C is a list of customer locations")
        print("    given as pairs (x, y) of integer coordinates on or near")
        print("    existing edges E. To integrate a set of customer locations")
        print("    into a given map M = (V, E), use addTarget(M, C)")
        print("")


        seed = 0

    if seed==0:              # generate very simple test data
        nodes = 20           # number of points in map
        customers = 5        # number of  customers
        grid = True
        scale = False

    else:
        grid = False
        scale = True

    random.seed(seed)

    M = generateGraph(nodes, 4500)

    C = generateRandomTargets(M, customers)

    if log:
        print(f"Generated map with {nodes:d} nodes and "
              f"{customers:d} customer locations")
```

```
    if plot:
        label="" if seed==0 else f"seed={seed:4d}"
        plotMap(M, T=C, scale=scale, text=label, grid=grid)

    return M, C
```

Data Generation is reproducible

```
[23]: D1 = generateData(1234)
      D2 = generateData(1234)
      D1 == D2
```

[23]: True

2 Generating Data

This section demonstrates how you can generate the test data for the problem.

2.1 General Help Message

If you use `generateData()` without any parameters you will get a general help message.

```
[24]: M, C = generateData()
```

```
Usage:  M, C = generateData(seed=None,
                            nodes=35, customers=100,
                            plot=False, log=False)

   seed  the seed value to be used for data generation.
         To test the application use seed=0, it will create
         a small map, with a very few customer locations.

   nodes the number of intersections (nodes, vertices) in the generated map

   customers  the number of customers generated on the map

   log   Controls print output during data generation.

   plot  Controls graphical output during data generation.

Returns:

   M = (V, E) is the generated map given as a graph
     where V is a list of vertices, with each vertice
     given as a pair (x, y) of integer coordinates,
     and E is a list of edges, with each edge given
     as a pair (A, B) of vertices, with each vertex again
     given as a pair (x, y) of integer coordinates
```
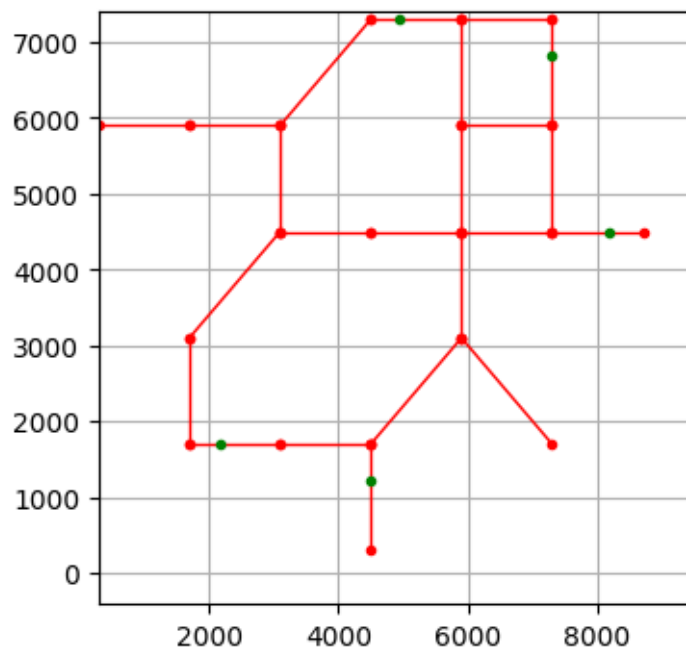
> C is a list of customer locations
>   given as pairs (x, y) of integer coordinates on or near
>   existing edges E. To integrate a set of customer locations
>   into a given map M = (V, E), use addTarget(M, C)

2.2 Simple Test Data

This section illustrates the data structure generated.

```
[25]: simpleData = generateData(seed=0, log=True, plot=True)
```

Generated map with 20 nodes and 5 customer locations


```
[26]: import pickle
      with open('simpleData.pickled', 'wb') as f:
          pickle.dump(simpleData, f)
```

```
[27]: M, C = simpleData
```

2.2.1 The Graph

You can identify the points in the grid above. The vertices of the graph are:

```
[28]: V, E = M
      V
```

```
[28]: [(300, 5900),
       (1700, 1700),
       (1700, 3100),
       (1700, 5900),
       (3100, 1700),
       (3100, 4500),
       (3100, 5900),
       (4500, 300),
       (4500, 1700),
       (4500, 4500),
       (4500, 7300),
       (5900, 3100),
       (5900, 4500),
       (5900, 5900),
       (5900, 7300),
       (7300, 1700),
       (7300, 4500),
       (7300, 5900),
       (7300, 7300),
       (8700, 4500)]
```

The edges of the graph are:

```
[29]: E
```

```
[29]: [((300, 5900), (1700, 5900)),
       ((1700, 3100), (1700, 1700)),
       ((3100, 1700), (1700, 1700)),
       ((3100, 4500), (1700, 3100)),
       ((3100, 4500), (4500, 4500)),
       ((3100, 5900), (1700, 5900)),
       ((3100, 5900), (3100, 4500)),
       ((3100, 5900), (4500, 7300)),
       ((4500, 1700), (3100, 1700)),
       ((4500, 1700), (4500, 300)),
       ((5900, 3100), (4500, 1700)),
       ((5900, 3100), (5900, 4500)),
       ((5900, 3100), (7300, 1700)),
       ((5900, 4500), (4500, 4500)),
       ((5900, 5900), (5900, 4500)),
       ((5900, 5900), (5900, 7300)),
       ((5900, 5900), (7300, 5900)),
       ((5900, 7300), (4500, 7300)),
       ((7300, 4500), (5900, 4500)),
       ((7300, 4500), (8700, 4500)),
       ((7300, 5900), (7300, 4500)),
       ((7300, 5900), (7300, 7300)),
       ((7300, 7300), (5900, 7300))]
```

2.2.2 Customer Addresses

The customer addresses (green dots in the map) are:
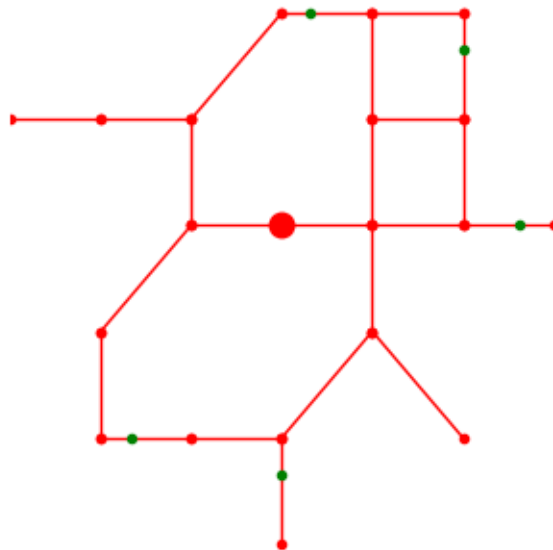
```
[30]: C
```

```
[30]: [(2176, 1700), (4500, 1224), (4929, 7300), (7300, 6825), (8167, 4500)]
```

2.2.3 Warehouse Location

The (default) Warehouse should be located near the centre of the map:

```
[31]: W = generateWarehouseLocation(M)
```
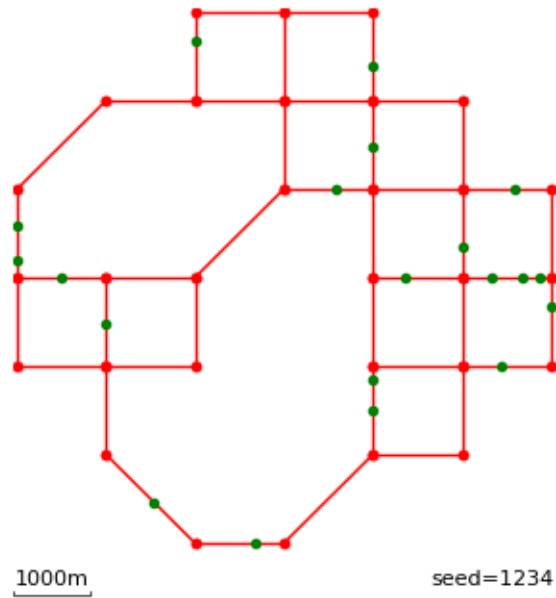
```
[32]: plotMap((V, E), T=C, w=W)
```


2.3 Test Data

During development of your simulation study it may be handy to uses simple test data, as this saves some compute time.

```
[35]: testData = generateData(1234, nodes=30, customers=20, plot=True, log=True)
```

```
Generated map with 30 nodes and 20 customer locations
```

1000m   seed=1234
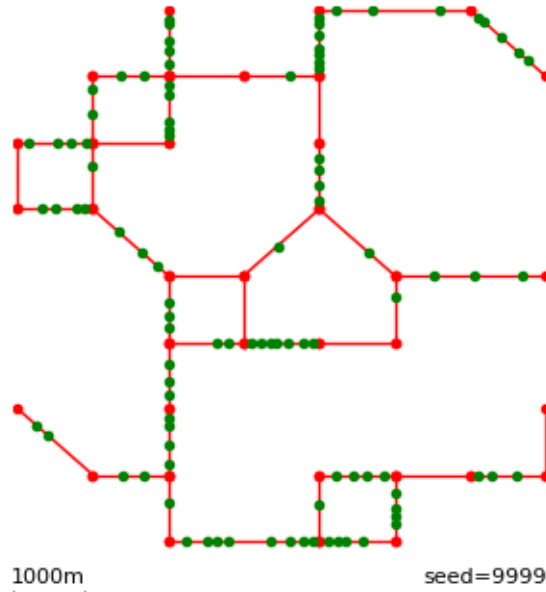
```
[36]: import pickle
      with open('testData.pickled', 'wb') as f:
          pickle.dump(testData, f)
```

2.4 Real Sample Data

This section shows sample data as you you may generate them for your required simulation.

```
[33]: data = generateData(9999, plot=True, log=True)
```

```
Generated map with 35 nodes and 100 customer locations
```

1000m   seed=9999

Save sample data as pickle file:

```
[34]: import pickle
      with open('data.pickled', 'wb') as f:
          pickle.dump(data, f)
```

Note: This notebook generates datafiles: `simpleData.pickled`, `data.pickled`, and `testData.pickled`. If you have changed parameters resulting in different data files, copy the newly generated data files over into the other subdirectories of this project. The downloadable version contains copies of the original data files.

2.5   Generating Delivery Data

This code is identical to the method generateDeliveries() used during the project.

```
[ ]: def generateDeliveryData(p, C, days, seed=0):
         ## p is the average number of parcels per day per customer
         ## C is the number of customers to be served
         ## days is the number of days for which data are to be generated.
         random.seed(seed)
         deliveries = [ [ ] for _ in range(days) ]
         for c in range(C):
             arr = 0
             while True:
                 arr += random.expovariate(p)
                 day = int(arr)
                 if day>=days:
                     break
```

21

```
        deliveries[day].append(c)
    return deliveries
```

2.6 Generate Warehouse Locations

During the project we used for simplicity a central warehouse location, generated by the method generateWarehouseLocation():

```
[20]: from statistics import median

      def generateWarehouseLocation(M):
          V, _ = M
          xc = median([ x for (x, y) in V ])
          yc = median([ y for (x, y) in V ])
          cloc = (xc, yc)
          minloc = V[0]
          mindist = dist(minloc, cloc)
          for i in range(1, len(V)):
              d = dist(V[i], cloc)
              if d<mindist:
                  minloc = V[i]
                  mindist = dist(V[i], cloc)
          return minloc
```

The CA specification refers to a set of warehouse locations. Use the method `generateMultipleWareHouseLocations(M)` (implying the default value N=5) to generate the locations for the map you have created using the last 4 digits of your student number. The last location on this list is always the central warehouose location we used during the project.

```
[ ]: def generateMultipleWarehouseLocations(M, N=5):
         C = generateWarehouseLocation(M)
         V, _ = M
         distances = [ (i, pathLength(shortestPath(M, V[i], C))) for i in␣
      ↪range(len(V)) ]
         faraway = sorted(distances, key=lambda x: x[1], reverse=True)
         index = [ i*(len(faraway)-1)//(N-1) for i in range(N)]
         locations = [ V[faraway[i][0]] for i in index ]
         return locations
```