3-Earth_Movers_Distance-Hungarian

February 15, 2018

In [1]:
```
import numpy as np
from scipy.spatial import distance_matrix
from scipy.optimize import linear_sum_assignment
from ipywidgets import FloatProgress
from IPython.display import display
```

In [2]:
```
import pickle
def load_object(filename):
    with open(filename, 'rb') as f:
        return pickle.load(f)
```

In [3]:
```
point_cloud_0 = np.array(load_object("../teapot.cloud"))
point_cloud_1 = np.array(load_object("../violin_case.cloud"))
np.random.shuffle(point_cloud_0)
np.random.shuffle(point_cloud_1)
point_cloud_0 = point_cloud_0[:10]
point_cloud_1 = point_cloud_1[:10]
D = distance_matrix(point_cloud_0, point_cloud_1)
```

In [4]:
```
# Source: https://en.wikipedia.org/wiki/Hungarian_algorithm
def hungarian(cost_matrix):
    # Copy cost matrix
    cost_matrix = np.copy(cost_matrix)
    N = cost_matrix.shape[0]

    # Subtract row minima
    for row in range(N):
        cost_matrix[row, :] -= np.min(cost_matrix[row, :])

    # Subtract column minima
    for col in range(N):
        cost_matrix[:, col] -= np.min(cost_matrix[:, col])

    progress = FloatProgress(min=0, max=N); display(progress)
    #for _ in range(3):
    while True:
        #print("Cost")
        #print(cost_matrix)
```

```
# Find valid row and column assignments
assigned_rows = np.full(N, False, dtype=np.bool)
assigned_columns = np.full(N, False, dtype=np.bool)
zeros_status = np.full((N, N), 0, dtype=np.int8) # 0 = unassigned, 1 = assigne

# For every row, if the row contains only one 0, assign it
for row in range(N):
    column_indices = np.argwhere(np.isclose(cost_matrix[row, :], 0))
    if len(column_indices) == 1: # Only one zero
        col = column_indices[0][0]
        if not assigned_columns[col]: # If column not assigned already
            # Assign column and row
            assigned_columns[col] = True
            assigned_rows[row] = True

            # Cancel other zeroes in the column
            zeros_status[np.where(np.isclose(cost_matrix[:, col], 0)), col] =
            zeros_status[row, col] = 1

# For every column, if the column contains only one 0, assign it
for col in range(N):
    row_indices = np.argwhere(np.isclose(cost_matrix[:, col], 0))
    if len(row_indices) == 1: # Only one zero
        row = row_indices[0][0]
        if not assigned_rows[row] and not assigned_columns[col]: # If row and
            # Assign column and row
            assigned_columns[col] = True
            assigned_rows[row] = True

            # Cancel other zeroes in the row
            zeros_status[row, np.where(np.isclose(cost_matrix[row, :], 0))] =
            zeros_status[row, col] = 1

#print("Zeros")
#print(zeros_status)

# Check for single zeroes after cancelation
for row in range(N):
    if not assigned_rows[row]:
        column_index = -1
        for col in range(N):
            if np.isclose(cost_matrix[row, col], 0) and zeros_status[row, col]
                if column_index == -1:
                    column_index = col
                else:
                    column_index = -1
                    break
```

```
            if column_index != -1:# and not assigned_columns[column_index]:
                # Assign column and row
                assigned_columns[column_index] = True
                assigned_rows[row] = True
                zeros_status[row, column_index] = 1

    for col in range(N):
        if not assigned_columns[col]:
            row_index = -1
            for row in range(N):
                if np.isclose(cost_matrix[row, col], 0) and zeros_status[row, col]
                    if row_index == -1:
                        row_index = row
                    else:
                        row_index = -1
                        break

            if row_index != -1:# and not assigned_rows[row_index]:
                # Assign column and row
                assigned_columns[col] = True
                assigned_rows[row_index] = True
                zeros_status[row_index, col] = 1

    #print("#Assignments")
    #print(np.sum(zeros_status == 1))

    # If number of assignments == N, we have reached the optimal solution
    num_assignments = np.sum(zeros_status == 1)
    progress.value = num_assignments
    progress.description = "%i/%i" % (num_assignments, N)
    if num_assignments == N:
        return list(range(N)), [np.argmax(zeros_status[row]) for row in range(N)]

    # Create lines by marking rows and columns
    marked_rows = np.full(N, False, dtype=np.bool)
    marked_columns = np.full(N, False, dtype=np.bool)
    for row in range(cost_matrix.shape[0]):
        # For every row without assignment
        if not assigned_rows[row]:
            # Mark the row
            marked_rows[row] = True

            # Find the zeros in the row
            for col in range(cost_matrix.shape[1]):
                if np.isclose(cost_matrix[row, col], 0):
                    # Mark the column
                    marked_columns[col] = True
```

```
                                # Find row with assignment in the current column
                                for row2 in range(cost_matrix.shape[0]):
                                    if zeros_status[row2, col] == 1:
                                        marked_rows[row2] = True

                    lines = np.full((N, N), 0, dtype=np.uint8)
                    for row in range(cost_matrix.shape[0]):
                        for col in range(cost_matrix.shape[1]):
                            if not marked_rows[row]:
                                lines[row, col] += 1
                            if marked_columns[col]:
                                lines[row, col] += 1

                    #print("Lines")
                    #print(lines)

                    min_uncovered = np.min(cost_matrix[np.where(lines == 0)])
                    cost_matrix[np.where(lines == 0)] -= min_uncovered # subtract uncovered values
                    cost_matrix[np.where(lines == 2)] += min_uncovered # add to intersections


            row_ind, col_ind = linear_sum_assignment(D)
            print("linear_sum_assignment(D) ->")
            print(D[row_ind, col_ind].sum())

            row_ind, col_ind = hungarian(D)
            print("hungarian(D) ->")
            print(D[row_ind, col_ind].sum())

linear_sum_assignment(D) ->
557.3235828799391


A Jupyter Widget


hungarian(D) ->
556.5343843579753
```

Unfortunately it gets stuck on bigger examples, but here I show it works when I sample 10 points of the point clouds.