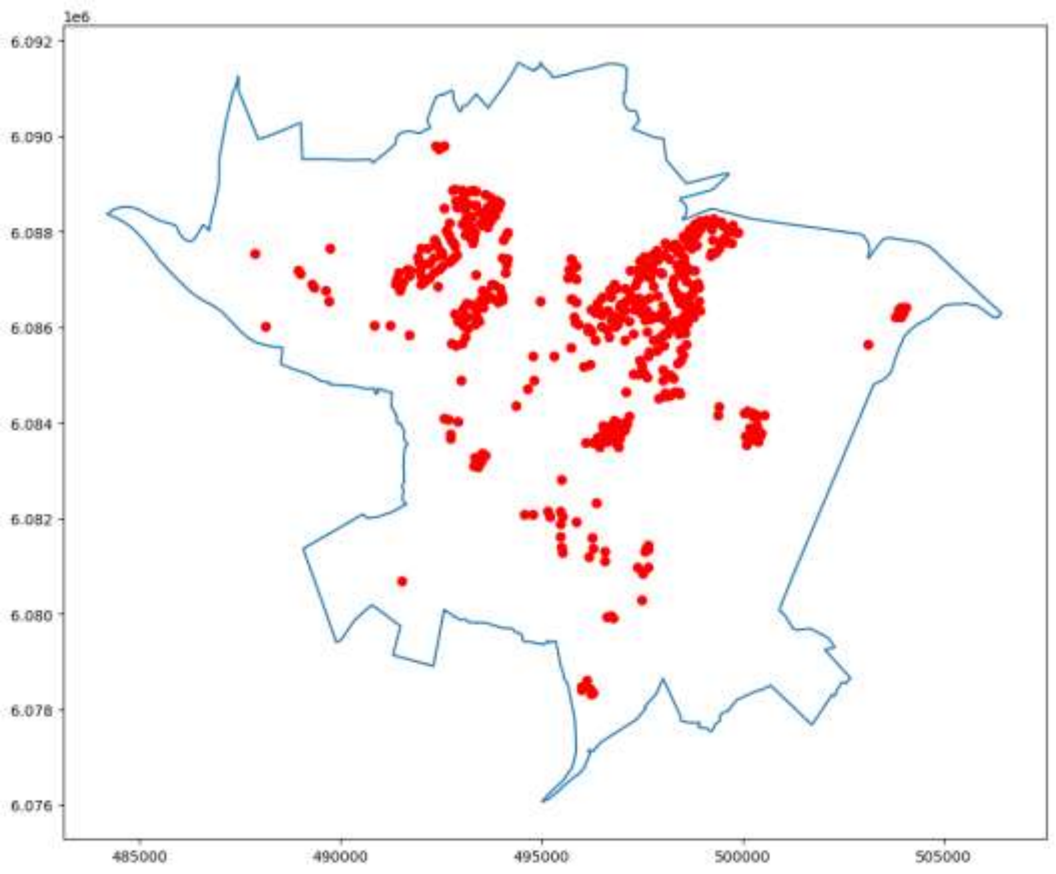
NUMERICAL METHODS AND ALGORITHMS

ZAHI EL HELOU
VARUN JOSH VIMALRAJ
TAMEEM ANSARI MAHADEER ALI
ABDELRHMAN IBRAHIM
ZALAL YOUSSEF

OPTIMIZATION

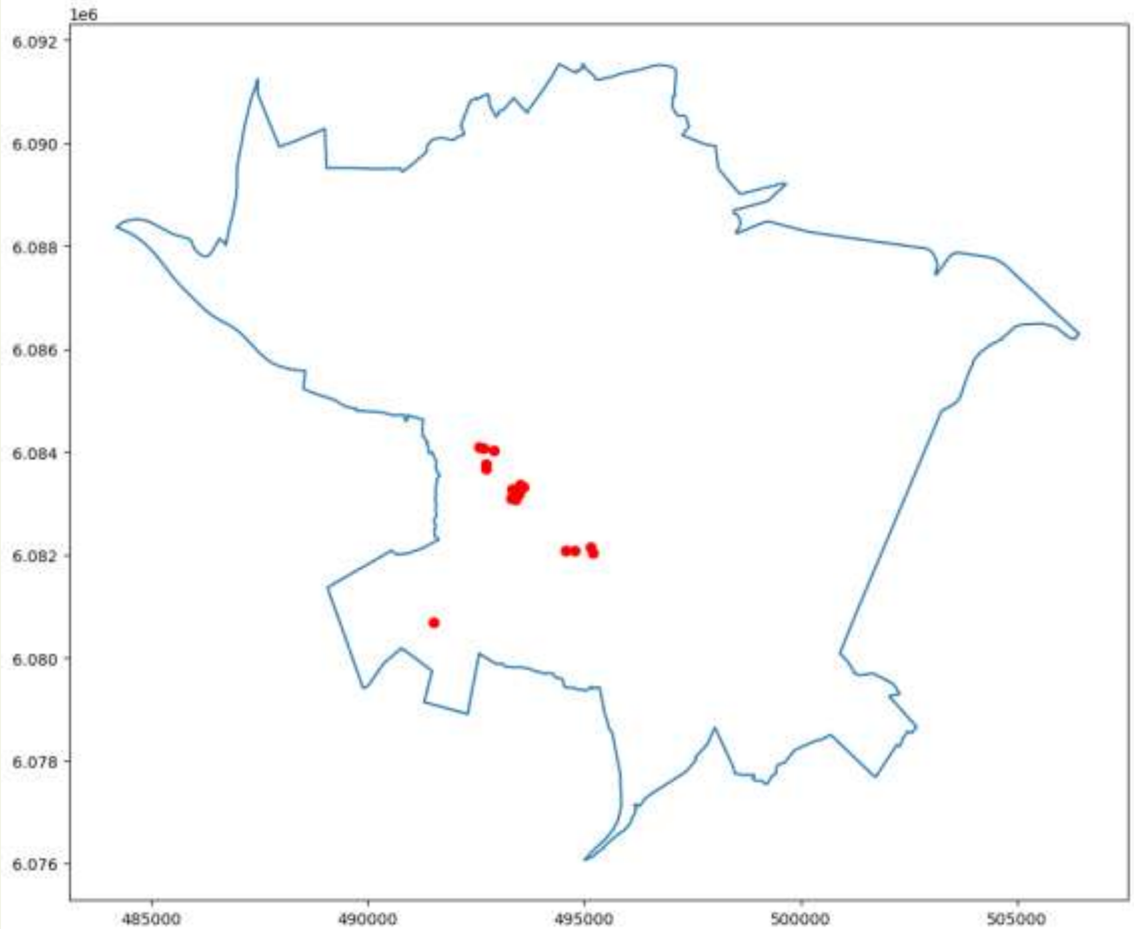
Task:

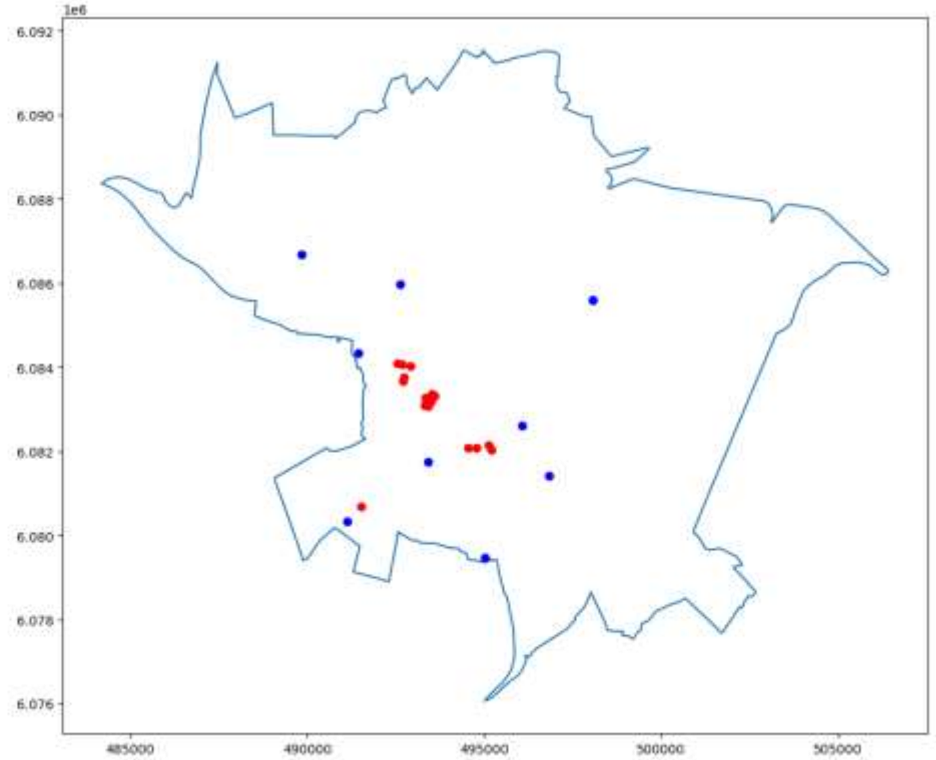
Help city policy makers to decide on where 10 new recycling containers should be installed. Create the target function and find the best coordinates for the new containers using gradient optimization.



The recycle bins present in Kaunas

The chosen region Aleksoto Seniunija





10 RANDOM POINTS THAT NEEDS TO BE OPTIMIZED

THE FACTOR WE CHOOSE FOR OPTIMIZATION AND SELECTION OF POINTS IS POPULATION.

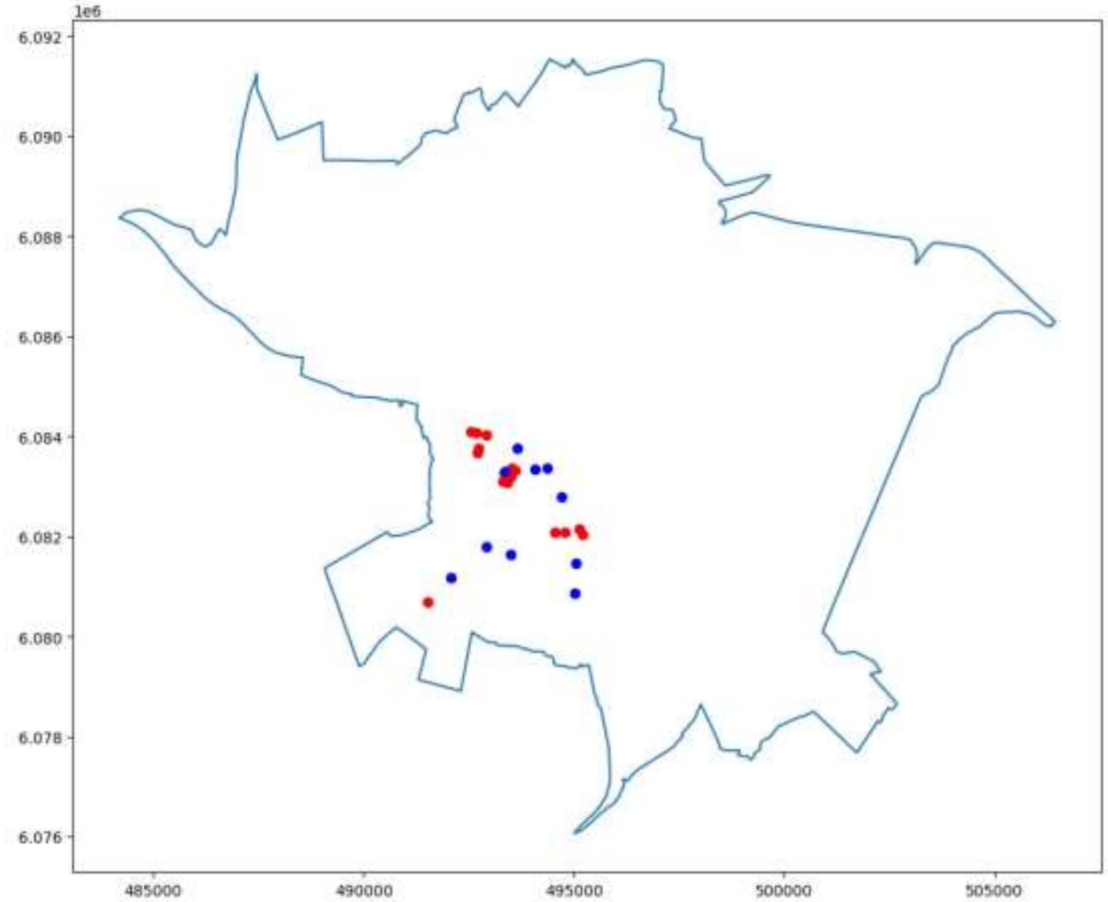
```python
def objectiveFunction(positionsGivenNodes, positionsOptNodes, pop1):
    min_pop1 = min(pop1)
    max_pop1 = max(pop1)
    normalized_pop1 = [(p - min_pop1) / (max_pop1 - min_pop1) for p in pop1]
    objFuncVal = 0
    avgDist = averageDistanceBetweenAllPoints(positionsGivenNodes, positionsOptNodes)

    for i in range(len(positionsGivenNodes)):
        for j in range(len(positionsOptNodes)):
            edgeDistance = distanceBetweenTwoPoints(positionsGivenNodes[i], positionsOptNodes[j])
            objFuncVal += ((avgDist - edgeDistance)**2)*normalized_pop1[j]

    for i in range(len(positionsOptNodes)):
        for j in range(i + 1, len(positionsOptNodes)):
            edgeDistance = distanceBetweenTwoPoints(positionsOptNodes[i], positionsOptNodes[j])
            objFuncVal += (((avgDist - edgeDistance))**2)*(normalized_pop1[i]+normalized_pop1[j])

    return objFuncVal
objVal = objectiveFunction(positionsGivenNodes, positionsOptNodes,pop1)
def quasiGradient(positionsGivenNodes, positionsOptNodes,pop1):
    h = 0.001
    f0 = objectiveFunction(positionsGivenNodes, positionsOptNodes,pop1)
    df = positionsOptNodes * 0;
    for i in range(0, len(positionsOptNodes)):
        for j in range (0,2):
            positionsOptNodesNew = positionsOptNodes;
            positionsOptNodesNew[i][j] += h
            f1 = objectiveFunction(positionsGivenNodes, positionsOptNodesNew,pop1)
            df[i][j] = (f1-f0)/h;
    return df
min_distance = 0.1
iter, step, eps = 0, 100, 1e-6
objValOld = objectiveFunction(positionsGivenNodes, positionsOptNodes,pop1)
print(objValOld)
grad = quasiGradient(positionsGivenNodes, positionsOptNodes,population)
while np.linalg.norm(grad[:,:]) > eps and iter < 1000 and step > 1e-6:
    grad = grad/np.linalg.norm(grad[:,:]);
    positionsOptNodes -= step * grad
    objValNew = objectiveFunction(positionsGivenNodes, positionsOptNodes,pop1)
    if objValOld < objValNew:
        positionsOptNodes += step * grad
        step = step * 0.9
    else:
        objValOld = objValNew
    for i in range(len(positionsOptNodes)):
        for j in range(i + 1, len(positionsOptNodes)):
            distance_ij = distanceBetweenTwoPoints(positionsOptNodes[i], positionsOptNodes[j])
            if distance_ij < min_distance:
                direction_vector = positionsOptNodes[j] - positionsOptNodes[i]
                direction_vector /= np.linalg.norm(direction_vector)
                positionsOptNodes[j] = positionsOptNodes[i] + min_distance * direction_vector

    grad = quasiGradient(positionsGivenNodes, positionsOptNodes,pop1)
    iter += 1
```

$$\min_{x_1,\dots,x_M,y_1,\dots,y_M} \Psi = \sum_{i=1}^{M} \sum_{j=i+1}^{N} \left((x_i - x_j)^2 + (y_i - y_j)^2 - \bar{d} \right)^2$$

ktu

1922

Optimized Result



THANK YOU

QUESTIONS?