ДОДАТОК А

Графічний матеріал атестаційної роботи

ГЮИК.504200.004

(позначення документу)

Харківський національний університет радіоелектроніки

«ЗАТВЕРДЖУЮ»
Керівник атестаційної роботи
проф. Безкоровайний В.В.

МЕТОД РЕІНЖИНІРИНГУ ТОПОЛОГІЧНИХ СТРУКТУР РЕГІОНАЛЬНИХ ЕЛЕКТРОЕНЕРГЕТИЧНИХ СИСТЕМ

Графічний матеріал

ЛИСТ ЗАТВЕРДЖЕННЯ

ГЮИК.504200.004-ЛЗ

РОЗРОБИВ:
ст. гр. СПРм-18-2
Губаренко М.С.

2020 р.

ЗАТВЕРДЖЕНО
ГЮИК.504200.004- ЛЗ

МЕТОД РЕІНЖИНІРИНГУ ТОПОЛОГІЧНИХ СТРУКТУР РЕГІОНАЛЬНИХ ЕЛЕКТРОЕНЕРГЕТИЧНИХ СИСТЕМ

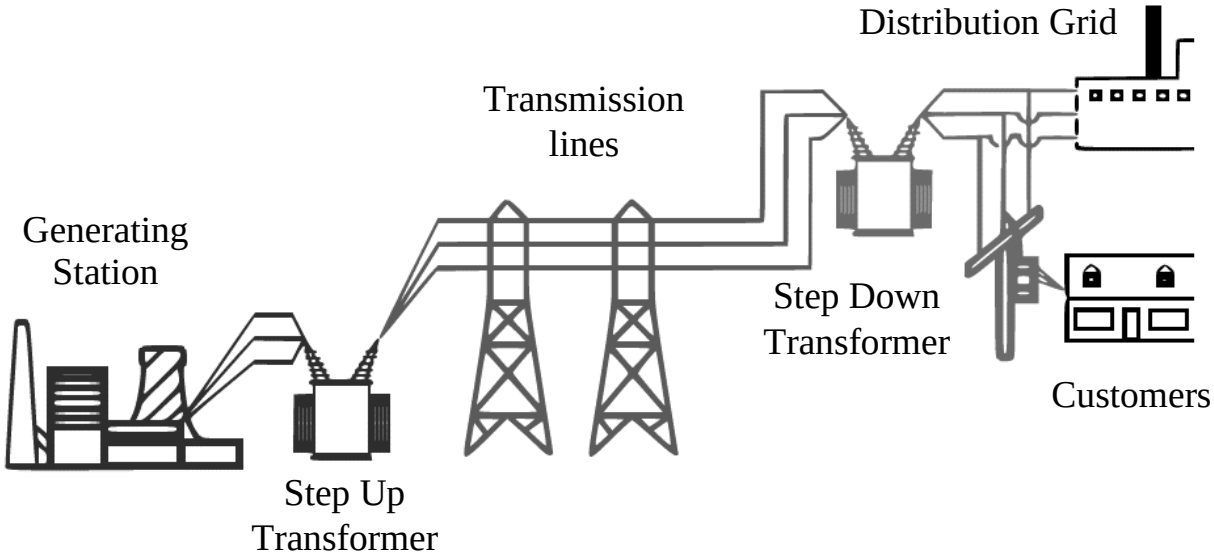
Графічний матеріал

ГЮИК.504200.004-ЛЗ

АРКУШІВ 8

2020 р.

СХЕМА ЕЛЕКТРОЕНЕРГЕТИЧНОЇ СИСТЕМИ



Generating Station – генератор.

Step Up Transformer – трансформаторна підстанція, що підвищує напругу.

Transmission lines – високовольтна магістраль.

Step Down Transformer – трансформаторна підстанція, що знижує напругу.

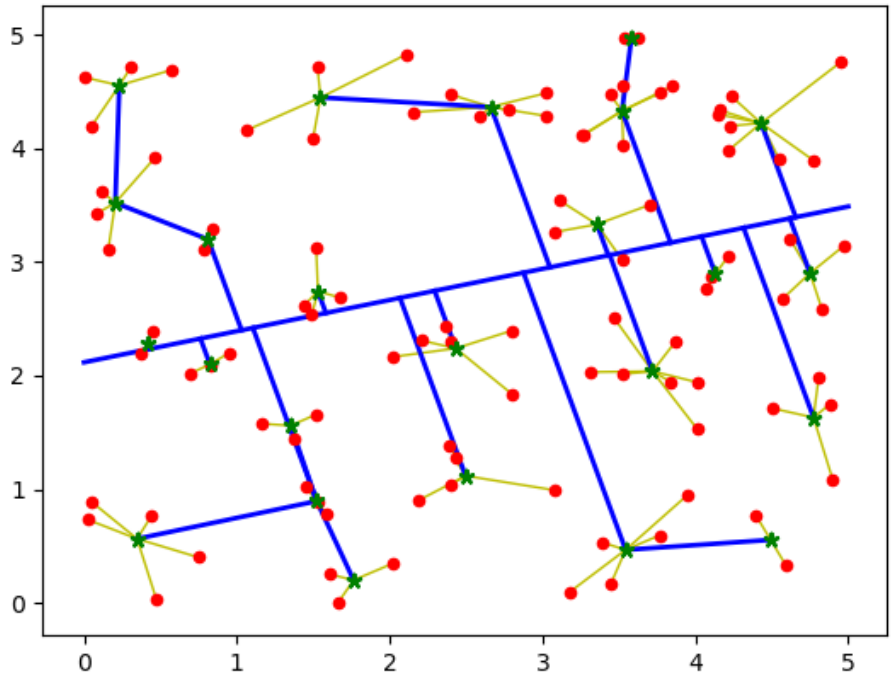
Distribution Grid – розподільна мережа.

Customers – споживачі.

Розробив	*Губаренко М.С.*			*Метод реінжинірингу топологічних структур регіональних електроенергетичних систем*	
Перевірив	*Безкоровайний В.В.*				
Н. контроль	*Безкоровайний В.В.*			*СПРм-18-2*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

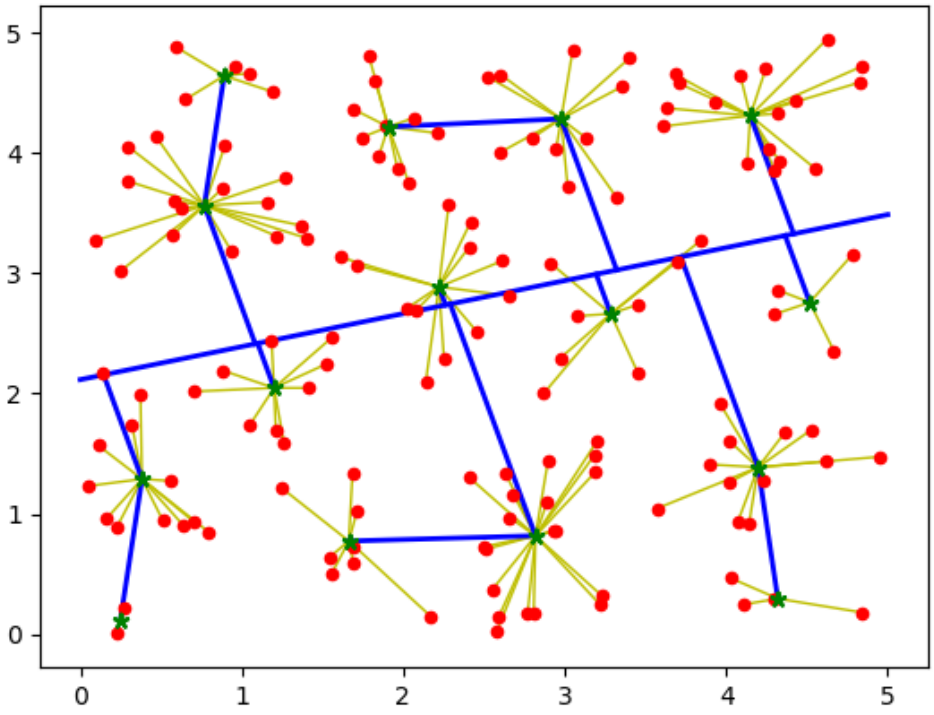
РЕЗУЛЬТАТ РЕІНЖИНІРИНГУ РОЗПОДІЛЬНОЇ МЕРЕЖІ ЕЕС

Початковий варіант мережі



C=21563,6 у.о.

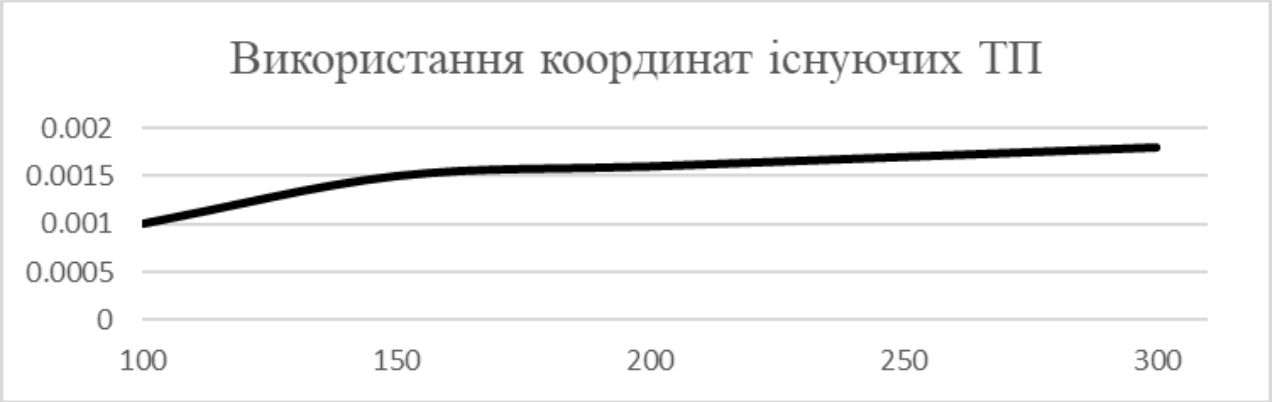
Варіант оптимізованої мережі



C=18492,3 у.о.

Розробив	*Губаренко М.С.*			*Метод реінжинірингу топологічних структур регіональних електроенергетичних систем*	
Перевірив	*Безкоровайний В.В.*				
Н. контроль	*Безкоровайний В.В.*			*СПРм-18-2*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

ЧАСОВА СКЛАДІСТЬ АЛГОРИТМУ ЗІ СТРАТЕГІЄЮ «КООРДИНАТИ ІСНУЮЧИХ ТП»

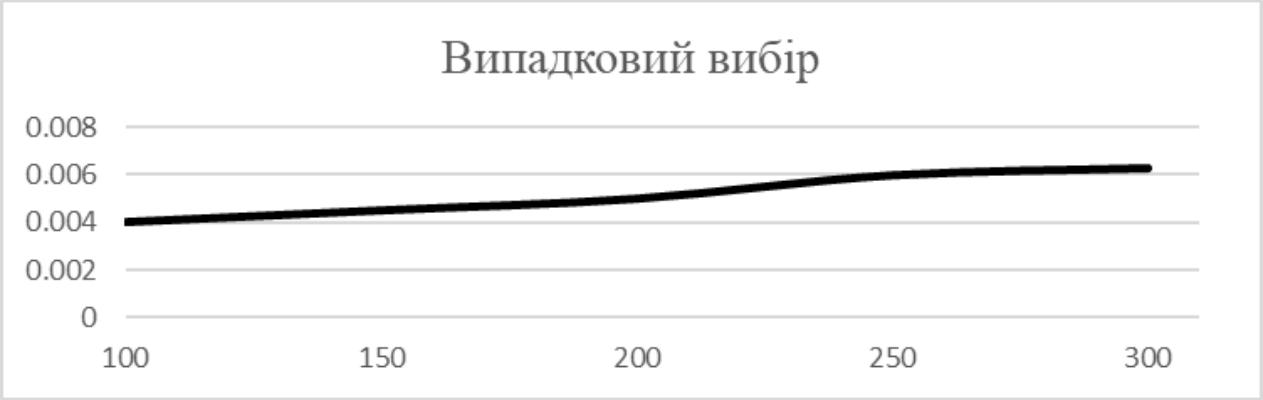


N	100	150	200	250	300
t, *сек*	0,001	0,0015	0,0016	0,0017	0,0018

Часова складність: $t(N) = -2 * 10^{-8} * N^2 + 10^{-5} * N + 4 * 10^{-17}$.

Розробив	*Губаренко М.С.*			*Метод реінжинірингу топологічних структур регіональних електроенергетичних систем*	
Перевірив	*Безкоровайний В.В.*				
Н. контроль	*Безкоровайний В.В.*			*СПРм-18-2*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

ЧАСОВА СКЛАДІСТЬ АЛГОРИТМУ ЗІ СТРАТЕГІЄЮ «ВИПАДКОВОГО ВИБОРУ»

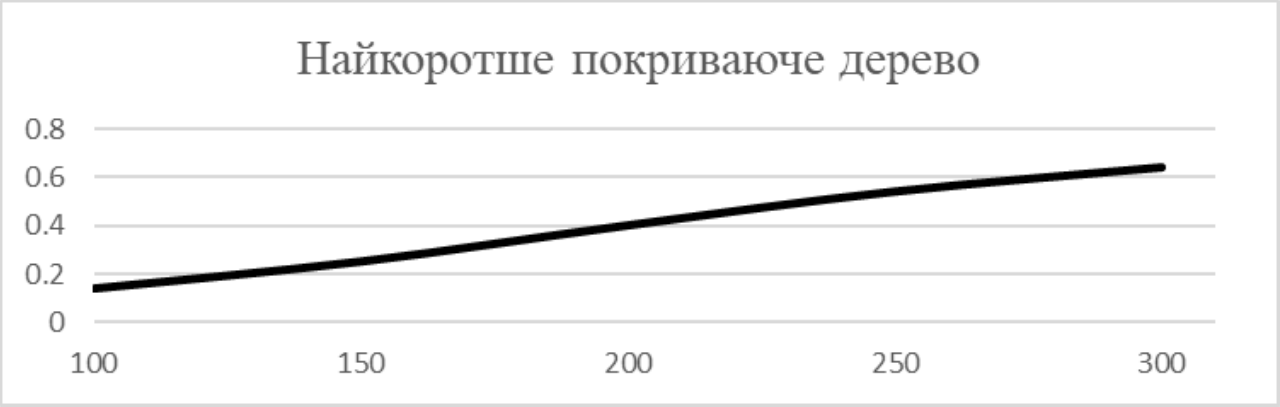


N	100	150	200	250	300
t, сек	0,004	0,0045	0,005	0,006	0,0063

Часова складність: $t(N) = 3 * 10^{-9} * N^2 + 10^{-5} * N + 0.0028$.

Розробив	*Губаренко М.С.*			*Метод реінжинірингу топологічних структур регіональних електроенергетичних систем*	
Перевірив	*Безкоровайний В.В.*				
Н. контроль	*Безкоровайний В.В.*			*СПРм-18-2*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

ЧАСОВА СКЛАДІСТЬ АЛГОРИТМУ ЗІ СТРАТЕГІЄЮ «НАЙКОРОТШЕ ПОКРИВАЮЧЕ ДЕРЕВО»

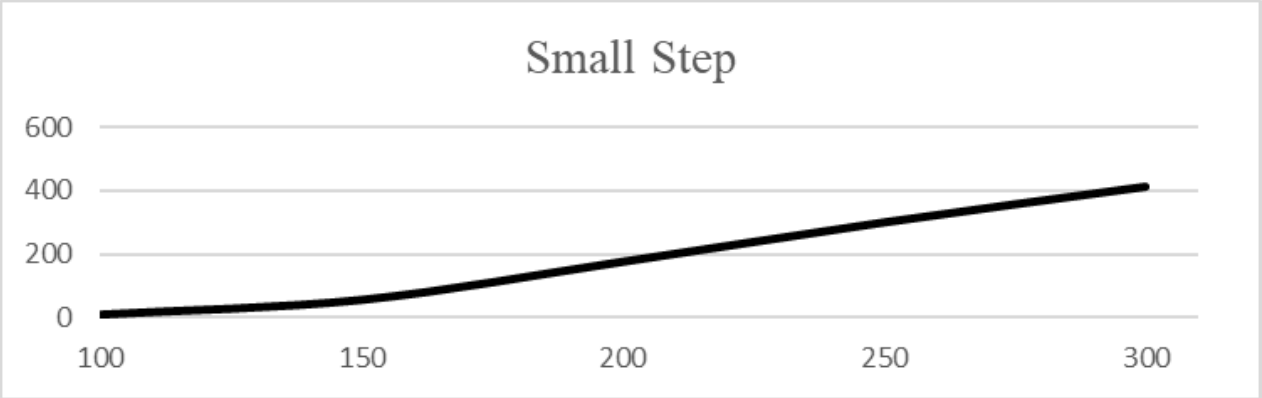


N	100	150	200	250	300
t, *сек*	0,138	0,25	0,4	0,54	0,64

Часова складність: $t(N) = -1 * 10^{-6} * N^2 + 0.003 * N + 0.158$.

Розробив	*Губаренко М.С.*			*Метод реінжинірингу топологічних структур регіональних електроенергетичних систем*	
Перевірив	*Безкоровайний В.В.*				
Н. контроль	*Безкоровайний В.В.*			*СПРм-18-2*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

ЧАСОВА СКЛАДІСТЬ АЛГОРИТМУ ЗІ СТРАТЕГІЄЮ «SMALL STEP»

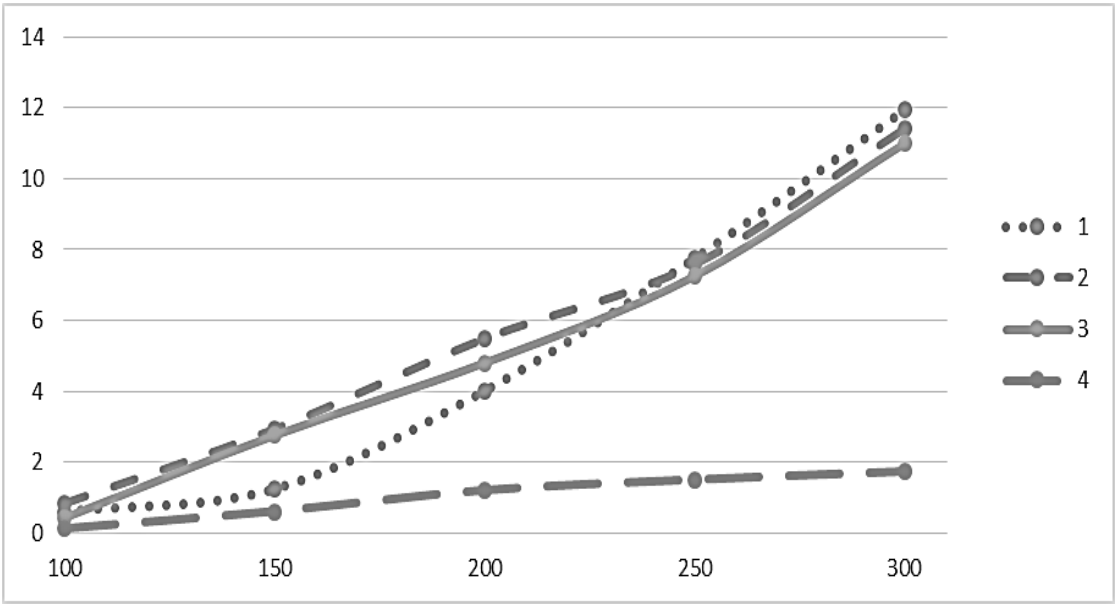


N	100	150	200	250	300
t, сек	11,83	58,18	177,4	300,5	411,76

Часова складність: $t(N) = 0.0038 * N^2 + 0.5564 * N - 91.238$.

Розробив	*Губаренко М.С.*			*Метод реінжинірингу топологічних структур регіональних електроенергетичних систем*	
Перевірив	*Безкоровайний В.В.*				
Н. контроль	*Безкоровайний В.В.*			*СПРм-18-2*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

ЧАСОВА СКЛАДІСТЬ РОБОТИ K-MEANS ПРИ РІЗНИХ СТРАТЕГІЯХ



1 – «Випадковий вибір»

Часова складність: $t(N) = 0.0002 * N^2 - 0.0343 * N + 1.535.$

2 – «Координати існуючих ТП»

Часова складність: $t(N) = 9 * 10^{-5} * N^2 + 0.0172 * N - 1.66.$

3 – «Найкоротше покриваюче дерево»

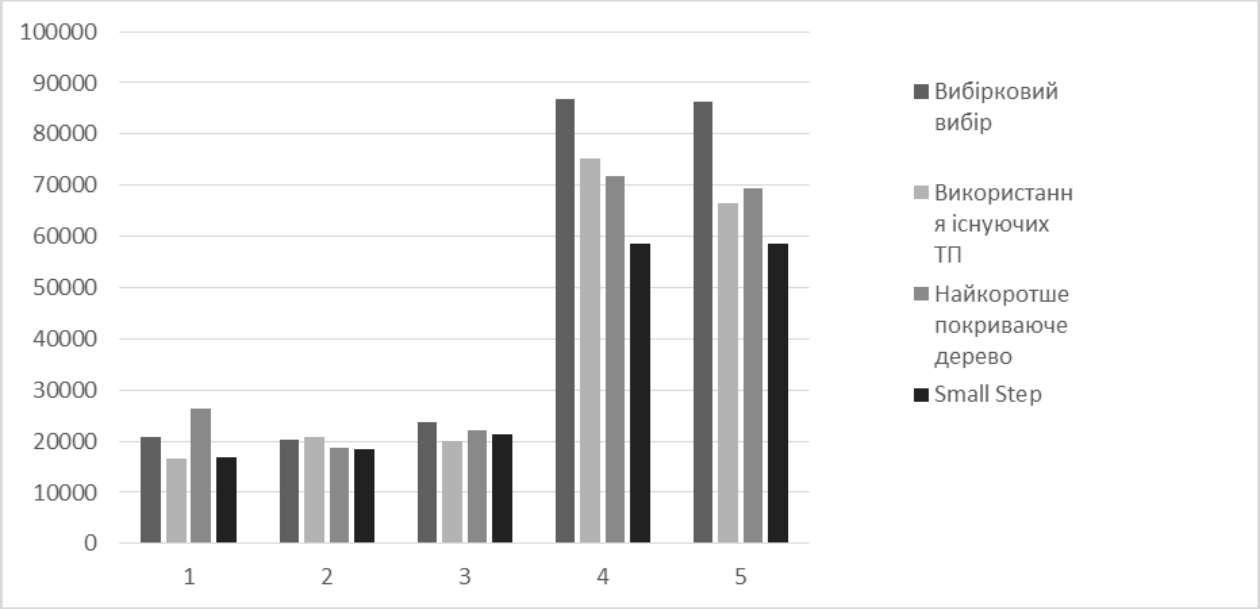
Часова складність: $t(N) = 9 * 10^{-5} * N^2 - 0.0143 * N - 1.78.$

4 – «Small Step»

Часова складність: $t(N) = -2 * 10^{-5} * N^2 - 0.0175 * N - 1.43.$

Кількість споживачів	100	150	200	250	300
1	20876,2	20762,1	20057,3	75261,3	66461,6
2	16522,5	20385,5	23799,2	86936,5	86259,3
3	26279,0	18650,6	22242,8	71832,9	69507,5
4	16781,8	18492,3	21348,5	58508,1	58509,8

ГІСТОГРАМА ПОРІВНЯЛЬНОГО АНАЛІЗУ ВАРТОСТІ РЕІНЖИНІРИНГУ МЕРЕЖІ



Кількість споживачів	100	150	200	250	300
«Випадковий вибір»	20876,2	20762,1	20057,3	75261,3	66461,6
«Координати існуючих ТП»	16522,5	20385,5	23799,2	86936,5	86259,3
«Найкоротше покриваюче дерево»	26279,0	18650,6	22242,8	71832,9	69507,5
«Small Step»	16781,8	18492,3	21348,5	58508,1	58509,8

Розробив	*Губаренко М.С.*			*Метод реінжинірингу топологічних структур регіональних електроенергетичних систем*	
Перевірив	*Безкоровайний В.В.*				
Н. контроль	*Безкоровайний В.В.*			*СПРм-18-2*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

ДОДАТОК Б

Текст програми

ГЮИК.504200.004-01 12 01

(позначення документу)

Харківський національний університет радіоелектроніки

«ЗАТВЕРДЖУЮ»
керівник атестаційної роботи
проф. Безкоровайний В.В.

МЕТОД РЕІНЖИНІРИНГУ ТОПОЛОГІЧНИХ СТРУКТУР РЕГІОНАЛЬНИХ ЕЛЕКТРОЕНЕРГЕТИЧНИХ СИСТЕМ

MODIFICATION K-MEANS
Текст програми

ЛИСТ ЗАТВЕРДЖЕННЯ

ГЮИК.504200.004-01 12 01-ЛЗ

РОЗРОБИВ:
ст. гр. СПРм-18-2
Губаренко М.С.

2020 р

ЗАТВЕРДЖЕНО
ГЮИК.504200.004-01 12 01-ЛЗ

МЕТОД РЕІНЖИНІРИНГУ ТОПОЛОГІЧНИХ СТРУКТУР РЕГІОНАЛЬНИХ ЕЛЕКТРОЕНЕРГЕТИЧНИХ СИСТЕМ

MODIFICATION K-MEANS

Текст програми

ГЮИК.504200.004-01 12 01

АРКУШІВ 10

2020 р

Файл k_means

```
import pandas as pd
import numpy as np
def points_generation(n_points, start_x, exd_x, start_y, end_y):
    points = pd.DataFrame({'X': np.random.uniform(start_x, exd_x,
n_points),
                           'Y': np.random.uniform(start_y, end_y,
n_points)})
    points.to_excel('Data/points.xlsx')
def centroids_generation(n_clusters, start_x, exd_x, start_y, end_y):
    centroids = pd.DataFrame({'X': np.random.uniform(start_x, exd_x,
n_clusters),
                              'Y': np.random.uniform(start_y, end_y,
n_clusters)})
    centroids.to_excel('Data/centroids.xlsx')
def evclid_distance(x1, y1, x2, y2):
    return pow(pow(x1 - x2, 2) + pow(y1 - y2, 2), 1 / 2)
def search_index(index, index_const_clusters):
    for i in range(len(index_const_clusters)):
        if index == index_const_clusters[i]:
            return True
    return False
def calculate_distance(list_points, list_clusters):
    n = list_clusters.index.size
    m = list_points.index.size
    distance = [[0] * m for i in range(n)]
    for i in range(n):
        for j in range(m):
            distance[i][j] = evclid_distance(list_points.at[j, 'X'],
list_points.at[j, 'Y'],
                                             list_clusters.at[i, 'X'],
list_clusters.at[i, 'Y'])
    return distance
def cluster_distribution(distances, max_distance, n, m):
    point_cluster = [[-1] * m for i in range(n)]
    for j in range(m):
        min_distance = distances[0][j]
        cluster_i = -1
        for i in range(n):
            if max_distance != -1:
                if distances[i][j] <= min_distance and distances[i][j]
<= max_distance:
                    min_distance = distances[i][j]
                    cluster_i = i
            else:
                if distances[i][j] <= min_distance:
                    min_distance = distances[i][j]
                    cluster_i = i
        if cluster_i != -1:
            point_cluster[cluster_i][j] = cluster_i
```

```
    return point_cluster
def recalculation_centroids(points, point_cluster, clusters,
index_const_clusters):
    n = clusters.index.size
    m = points.index.size
    for i in range(n):
        if search_index(i, index_const_clusters):
            continue
        mean_x = 0
        mean_y = 0
        count = 0
        for j in range(m):
            if point_cluster[i][j] == i:
                mean_x += points.at[j, 'X']
                mean_y += points.at[j, 'Y']
                count = count + 1
        if count != 0:
            clusters.at[i, 'X'] = mean_x / count
            clusters.at[i, 'Y'] = mean_y / count
def compare_array(array1, array2, n, m):
    for i in range(n):
        for j in range(m):
            if array1[i][j] != array2[i][j]:
                return False
    return True
def find_index_outside_point(point_cluster, n, m):
    point_j = -1
    for j in range(m):
        k = 0
        for i in range(n):
            if point_cluster[i][j] != -1:
                k = 1
        if k == 0:
            point_j = j
            break
    return point_j
def calculate_distance_points(list_points):
    m = list_points.index.size
    distance = [[0] * m for i in range(m)]
    for i in range(m):
        for j in range(m):
            distance[i][j] = evclid_distance(list_points.at[j, 'X'],
list_points.at[j, 'Y'],
                                             list_points.at[i, 'X'],
list_points.at[i, 'Y'])
    return distance
def calculate_distance_centroids(list_centroids):
    n = list_centroids.index.size
    distance = [[0] * n for i in range(n)]
    for i in range(n):
        for j in range(i + 1, n):
```

```
            distance[i][j] = evclid_distance(list_centroids.at[j,
'X'], list_centroids.at[j, 'Y'],
                                             list_centroids.at[i,
'X'], list_centroids.at[i, 'Y'])
            distance[j][i] = distance[i][j]
    return distance
def min_distance(list_distance, n, m):
    min_dist = -1
    min_i = -1
    min_j = -1
    for i in range(n):
        for j in range(0, m):
            if min_dist >= list_distance[i][j] > 0 or (min_dist == -1
and list_distance[i][j] > 0):
                min_dist = list_distance[i][j]
                min_i = i
                min_j = j
    return min_i, min_j
def delete_centroid(list_centroids, centroid1, centroid2,
list_const_centroid):
    x = (list_centroids.at[centroid1, 'X'] +
list_centroids.at[centroid2, 'X']) / 2
    y = (list_centroids.at[centroid1, 'Y'] +
list_centroids.at[centroid2, 'Y']) / 2
    list_centroids.at[centroid1, 'X'] = x
    list_centroids.at[centroid1, 'Y'] = y
    for i in range(len(list_const_centroid)):
        if list_const_centroid[i] >= centroid2:
            list_const_centroid[i] -= 1
    list_centroids =
pd.DataFrame(list_centroids.drop(index=[centroid2]))
    list_centroids.reset_index(drop=True, inplace=True)
    return list_centroids
```

Файл reengineering

```
import k_means
import graph
import smallStep
import transmission_lines
import matplotlib.pyplot as plt
import pandas as pd
import copy
import time
def draw_clustering(number_figure, list_points, list_clusters,
list_point_cluster=None, list_transmission_line=None):
    print(list_clusters)
    n = list_clusters.index.size
    m = list_points.index.size
    plt.figure(number_figure)
    for i in range(n):
```

```
        for j in range(m):
            if list_point_cluster[i][j] == i:
                plt.plot([list_points.at[j, 'X'], list_clusters.at[i,
'X']],
                         [list_points.at[j, 'Y'], list_clusters.at[i,
'Y']], 'y',
                         linewidth=1, zorder=0)
    for i in range(len(list_transmission_line)):
        plt.plot([list_transmission_line[i][0],
list_transmission_line[i][2]],
                 [list_transmission_line[i][1],
list_transmission_line[i][3]], color="b", linewidth=2,
                 zorder=0)
    plt.scatter(list_points['X'], list_points['Y'], c='red', s=20,
zorder=1)
    plt.scatter(list_clusters['X'], list_clusters['Y'], c='green',
s=30, marker=(5, 2), zorder=1)
def get_centroids(n_method, list_points, n_figure=None,
n_centroids=None):
    list_centroids = pd.DataFrame()
    if n_method == 1:
        k_means.centroids_generation(n_centroid, 0, 5, 0, 5)
        list_centroids = pd.read_excel('Data/centroids.xlsx',
index_col=0)
    elif n_method == 2:
        list_centroids = pd.read_excel('Data/Coordinate_TP.xlsx',
index_col=None)
    elif n_method == 3:
        weight = k_means.calculate_distance_points(list_points)
        list_centroids =
graph.minimal_spanning_tree_clustering(n_figures + 1, weight,
list_points)
    elif n_method == 4:
        list_centroids = smallStep.small_step(list_points,
max_distances, n_figure)
    return list_centroids
def find_list(list_for_find, list_find):
    for i in range(len(list_for_find)):
        if list_for_find[i][0] == list_find[0] and list_for_find[i][1]
== list_find[1] and \
                list_for_find[i][2] == list_find[2] and
list_for_find[i][3] == list_find[3]:
            return True
    return False
max_distances = 1
n_centroid = 30
n_points = 150
n_figures = 1
n_method = 1
points = pd.read_excel('Data/points.xlsx', index_col=0)
n_points = points.index.size
```

```
n_figures = 1
plt.scatter(points['X'], points['Y'], c='red', s=20, zorder=1)
n_figures += 1
existing_clusters =
copy.deepcopy(pd.read_excel('Data/Coordinate_TP.xlsx',
index_col=None))
index_const_existing_centroids = []
index_const_starting_centroids = []
plt.figure(n_figures)
distances = k_means.calculate_distance(points, existing_clusters)
point_cluster = [[-1] * n_points for i in
range(existing_clusters.index.size)]
existing_point_cluster = k_means.cluster_distribution(distances, -1,
existing_clusters.index.size, n_points)
plt.title("start_data")
transmission_line = pd.read_csv("Data/Transmission_line.csv",
index_col=None)
transmission_line = transmission_line.values.tolist()
existing_list_line =
transmission_lines.get_transmission_lines(transmission_line,
existing_clusters)
draw_clustering(n_figures, points, existing_clusters, point_cluster,
existing_list_line)
n_figures += 1
n_method = 1
while n_method < 5:
    start_time = time.time()
    if n_method == 4:
        start_centroids = get_centroids(n_method, points, n_figures)
    if n_method == 1:
        start_centroids = get_centroids(n_method, points,
n_centroids=n_centroid)
    else:
        start_centroids = get_centroids(n_method, points)
    end_time = time.time()
    print(n_method)
    print("set centroids %s" % (end_time - start_time))
    centroids = copy.deepcopy(start_centroids)
    n_centroid = centroids.index.size
    old_point_cluster = [[-1] * n_points for i in range(n_centroid)]
    old_list_centroids = copy.deepcopy(centroids)
    is_add_centroid = False
    is_delete_centroid = False
    is_finish = False
    start_time = time.time()
    while True:
        const_cluster = copy.deepcopy(index_const_starting_centroids)
        old_const_cluster =
copy.deepcopy(index_const_starting_centroids)
        while True:
            while True:
```

```
                distances = k_means.calculate_distance(points,
centroids)
                point_cluster =
k_means.cluster_distribution(distances, max_distances, n_centroid,
n_points)
                if k_means.compare_array(point_cluster,
old_point_cluster, n_centroid, n_points):
                    break
                k_means.recalculation_centroids(points, point_cluster,
centroids, const_cluster)
                old_point_cluster = point_cluster
            index_point_outside =
k_means.find_index_outside_point(point_cluster, n_centroid, n_points)
            if is_finish or (-1 == index_point_outside and
is_add_centroid):
                break
            if index_point_outside != -1 and not is_delete_centroid:
                is_add_centroid = True
                centroids.loc[n_centroid] = {'X':
points.at[index_point_outside, 'X'],
                                             'Y':
points.at[index_point_outside, 'Y']}
                old_point_cluster = [[-1] * n_points for i in
range(n_centroid)]
                n_centroid = n_centroid + 1
            if not is_add_centroid and not is_finish:
                if -1 == index_point_outside:
                    is_delete_centroid = True
                    distances_between_centroids =
k_means.calculate_distance_centroids(centroids)
                    while True:
                        i, j =
k_means.min_distance(distances_between_centroids, n_centroid,
n_centroid)
                        if transmission_lines.find_element_in_list(j,
const_cluster) is not None or \
transmission_lines.find_element_in_list(i, const_cluster) is not None:
                            distances_between_centroids[i][j] = -1
                            distances_between_centroids[j][i] = -1
                        else:
                            break
                    if i != -1 or j != -1:
                        old_list_centroids = copy.deepcopy(centroids)
                        old_const_cluster =
copy.deepcopy(const_cluster)
                        centroids = k_means.delete_centroid(centroids,
i, j, const_cluster)
                        n_centroid = n_centroid - 1
                        old_point_cluster = [[-1] * n_points for i in
range(n_centroid)]
                    else:
```

```
                        is_finish = True
                        break
                else:
                    centroids = copy.deepcopy(old_list_centroids)
                    const_cluster = copy.deepcopy(old_const_cluster)
                    n_centroid = n_centroid + 1
                    is_finish = True
        distance_centroids =
k_means.calculate_distance(existing_clusters, centroids)
        n = centroids.index.size
        m = existing_clusters.index.size
        while True:
            min_i, min_j = k_means.min_distance(distance_centroids, n,
m)
            if transmission_lines.find_element_in_list(min_j,
index_const_existing_centroids) is not None:
                distance_centroids[min_i][min_j] = -1
            else:
                break
        if distance_centroids[min_i][min_j] <= max_distances / 10 and
distance_centroids[min_i][min_j] != -1:
            dist = []
            for i in range(start_centroids.index.size):

dist.append(k_means.evclid_distance(existing_clusters.at[min_j, 'X'],
existing_clusters.at[min_j, 'Y'],

start_centroids.at[i, 'X'], start_centroids.at[i, 'Y']))
            min_d = -1
            index = -1
            for i in range(start_centroids.index.size):
                if min_d > dist[i] >= 0 or min_d == -1 and dist[i] >=
0 and \
                        transmission_lines.find_element_in_list(i,
index_const_starting_centroids) is None:
                    min_d = dist[i]
                    index = i
            start_centroids.at[index, 'X'] =
existing_clusters.at[min_j, 'X']
            start_centroids.at[index, 'Y'] =
existing_clusters.at[min_j, 'Y']
            index_const_existing_centroids.append(min_j)
            index_const_starting_centroids.append(index)
            centroids = copy.deepcopy(start_centroids)
            old_list_centroids = copy.deepcopy(centroids)
            n_centroid = centroids.index.size
            is_add_centroid = False
            is_delete_centroid = False
            is_finish = False
            old_point_cluster = [[-1] * n_points for i in
range(n_centroid)]
```

```
        else:
            print(centroids)
            print(index_const_existing_centroids)
            print(index_const_starting_centroids)
            print(const_cluster)
            list_line =
transmission_lines.get_transmission_lines(transmission_line,
centroids)
            draw_clustering(n_figures, points, centroids,
point_cluster, list_line)
            n_figures += 1
            break
    end_time = time.time()
    print("clustering %s" % (end_time - start_time))
    len_line = 0
    c_line = 1
    c_new_TP = 100
    c_modern_TP = 10
    c_transmission_line = 2
    total_c = 0
    for i in range(n_centroid):
        for j in range(n_points):
            if point_cluster[i][j] == i:
                if transmission_lines.find_element_in_list(i,
const_cluster) is not None:
                    index =
index_const_existing_centroids[transmission_lines.find_element_in_list
(i, const_cluster)]
                    if existing_point_cluster[index][j] == index:
                        continue
                else:
                    len_line +=
k_means.evclid_distance(centroids.at[i, 'X'], centroids.at[i, 'Y'],
                                                        points.at[i,
'X'], points.at[i, 'Y']) * c_line
    total_c += len_line
    total_c += (centroids.index.size - len(const_cluster)) * c_new_TP
    total_c += len(const_cluster) * c_modern_TP
    sum_transmission_line = 0
    for i in range(len(list_line)):
        if find_list(existing_list_line, list_line[i]):
            continue
        sum_transmission_line +=
k_means.evclid_distance(list_line[i][0], list_line[i][1],

list_line[i][2], list_line[i][3]) * c_transmission_line
    total_c += sum_transmission_line
    print(total_c)
    n_method += 1
    index_const_existing_centroids = []
    index_const_starting_centroids = []
```

```
plt.show()
```

Файл graph

```
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt


def create_edge_list(weight, n):
    edge_list = []
    for i in range(n):
        for j in range(n):
            edge = [i, j, weight[i][j]]
            edge_list.append(edge)
    return edge_list


def draw_graph(graph, points):
    positions = {}
    for i in range(points.index.size):
        positions.update({i: (points.at[i, 'X'], points.at[i, 'Y'])})
    nx.draw(graph, pos=positions, node_size=10,
nodelist=positions.keys(), with_labels=positions)


def minimum_spanning_tree(n_vertex, weight):
    g = nx.Graph()
    edge_list = create_edge_list(weight, n_vertex)
    for edge in edge_list:
        g.add_edge(edge[0], edge[1], weight=edge[2])
    tree = nx.minimum_spanning_tree(g)
    print(tree.edges(data=True))
    return tree


def delete_edge(tree):
    g = nx.Graph()
    sum = 0
    count = 0
    for (u, v, d) in tree.edges(data=True):
        sum += d['weight']
        count += 1
    max_weight = sum / count
    for (u, v, d) in tree.edges(data=True):
        if d['weight'] < max_weight:
            g.add_edge(u, v, weight=d)
    return g


def definition_clusters(graph):
```

```
    clusters = []
    edges = list(graph.edges)
    print(edges)
    i: int = 0
    while i < len(edges):
        vertexes = [edges[i][0], edges[i][1]]
        edges.remove((edges[i][0], edges[i][1]))
        j: int = 0
        while j < len(edges):
            if edges[j][0] in vertexes:
                vertexes.append(edges[j][1])
                edges.remove((edges[j][0], edges[j][1]))
                j = 0
            elif edges[j][1] in vertexes:
                vertexes.append(edges[j][0])
                edges.remove((edges[j][0], edges[j][1]))
                j = 0
            else:
                j = j + 1
        clusters.append(vertexes)
    print(len(clusters))
    print(clusters)
    return clusters
def calculate_centroids(clusters, points):
    centroids = pd.DataFrame(columns=['X', 'Y'])
    k = 0
    for item in clusters:
        mean_x = 0
        mean_y = 0
        for i in item:
            mean_x += points.at[i, 'X']
            mean_y += points.at[i, 'Y']
        x = mean_x / len(item)
        y = mean_y / len(item)
        centroids.loc[k] = {'X': x, 'Y': y}
        k += 1
    return centroids
def minimal_spanning_tree_clustering(n_figure, weight, points):
    n_points = points.index.size
    plt.figure(n_figure)
    tree = minimum_spanning_tree(n_points, weight)
    #draw_graph(tree, points)
    plt.figure(n_figure)
    tree = delete_edge(tree)
    # draw_graph(tree, points)
    clusters = definition_clusters(tree)
    centroids = calculate_centroids(clusters, points)
    return centroids
```

ДОДАТОК В

Сертифікат учасника конференції

СЕРТИФІКАТ УЧАСТІ В V МІЖНАРОДНІЙ НАУКОВО-ПРАКТИЧНІЙ КОНФЕРЕНЦІЇ «ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В КУЛЬТУРІ, МИСТЕЦТВІ, ОСВІТІ, НАУЦІ, ЕКОНОМІЦІ ТА БІЗНЕСІ»

ДОВІДКА ПРО УЧАСТЬ В ЗВІТНІЙ НАУКОВІЙ КОНФЕРЕНЦІЇ ВИКЛАДАЧІВ, ДОКТОРАНТІВ, АСПІРАНТІВ ТА СТУДЕНТІВ ПРИКАРПАТСЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ ІМ. В. СТЕФАНИКА

Міністерство освіти і науки України
Державний вищий навчальний заклад
Прикарпатський національний університет імені Василя Стефаника

вул. Шевченка, 57, м. Івано-Франківськ, 76018, тел. (0342) 75-23-51, факс (0342) 53-15-74
e-mail: office@pnu.edu.ua, Код ЄДРПОУ 02125266

19.05.2009 № 01-03/167

На № _____ від _____

Довідка

Видана Губаренко Марині Станіславівні, студентці II курсу ОР магістр факультету комп'ютерних наук Харківського національного університету радіоелектроніки, про те, що стаття «Декопозиція задачі реінжинірингу регіональних електроенергетичних систем» за її авторством прийнята до публікації у збірнику студентських наукових праць «Еврика - XXI» ДВНЗ «Прикарпатський національний університет імені Василя Стефаника»

Проректор В.М. Якубів

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ Державний вищий навчальний заклад Прикарпатський національний університет імені Василя Стефаника І.К. 02125266

ДОДАТОК Г

Відомість атестаційної роботи

ГЮИК.504200.004

(позначення документу)

№	Позначення	Найменування	Додаткові відомості
1	ГЮИК.504200.004 ПЗ	**Текстові документи**	
2	ГЮИК.504200.004 - 01 12 01	Пояснювальна записка	70 с.
		Текст програми	12 с.
3		**Графічні документи**	
		Схема електроенергетичної	
4		мережі	1 аркуш
		Результат реінжинірингу	
5		розподільної мережі ЕЕС	1 аркуш
		Часова складність алгоритму	
		зі стратегією «Координати	
6		існуючих ТП»	1 аркуш
		Часова складність	
		алгоритму зі стратегією	
7		«Випадкового вибору»	1 аркуш
		Часова складність	
		алгоритму зі стратегією	
		«Найкоротше покриваюче	
8		дерево»	1 аркуш
		Часова складність	
		алгоритму зі стратегією	
9		«Small Step»	1 аркуш
		Часова складність	
		алгоритму k-means при	
10		різних стратегіях	1 аркуш
		Гістограма порівняльної	
		вартості реінжинірингу	
		мережі	1 аркуш

Змін.	Арк.	№ документа	Підпис	Дата	ГЮИК.504200.004 ДЗ		
Розробив		Губаренко М.С.		20.05.20	Метод реінжинірингу топологічних структур регіональних електроенергетичних систем	Аркуш	Аркушів
Перевірив		Безкоровайний В.В.		20.05.20		1	1
Н. контроль		Безкоровайний В.В.		20.05.20		ХНУРЕ Кафедра СТ	
Затвердив		Гребеннік І.В.					