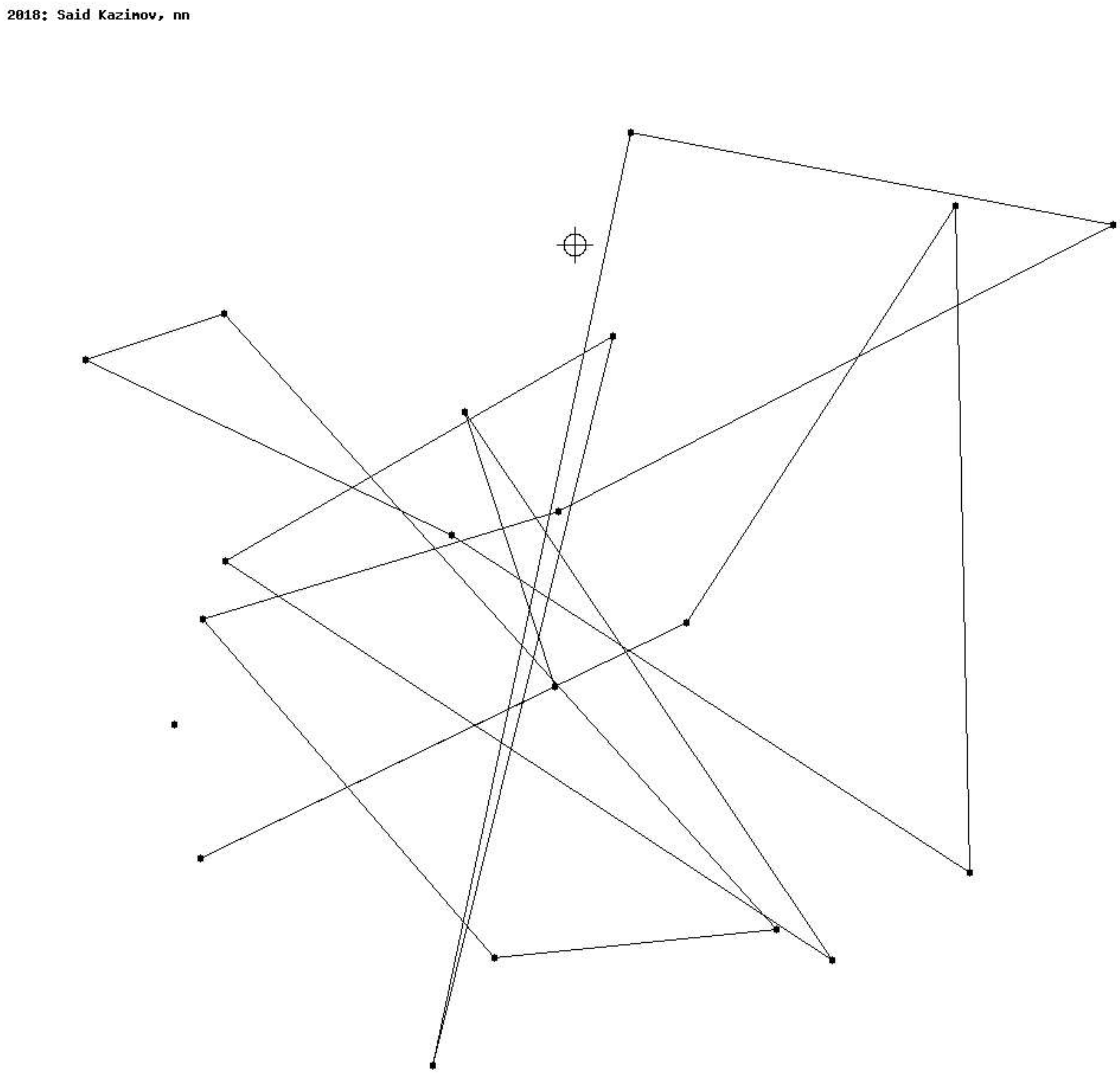
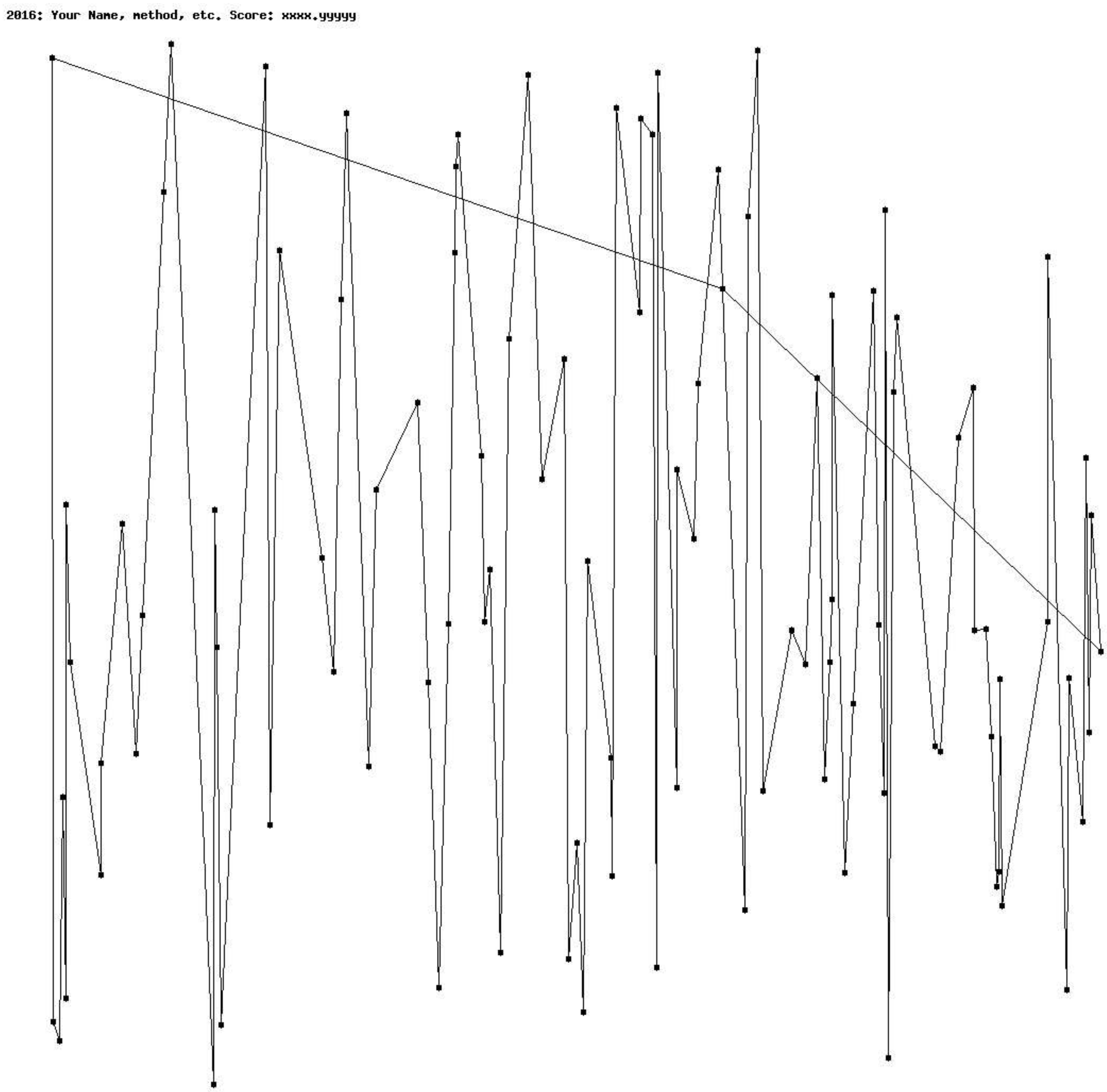
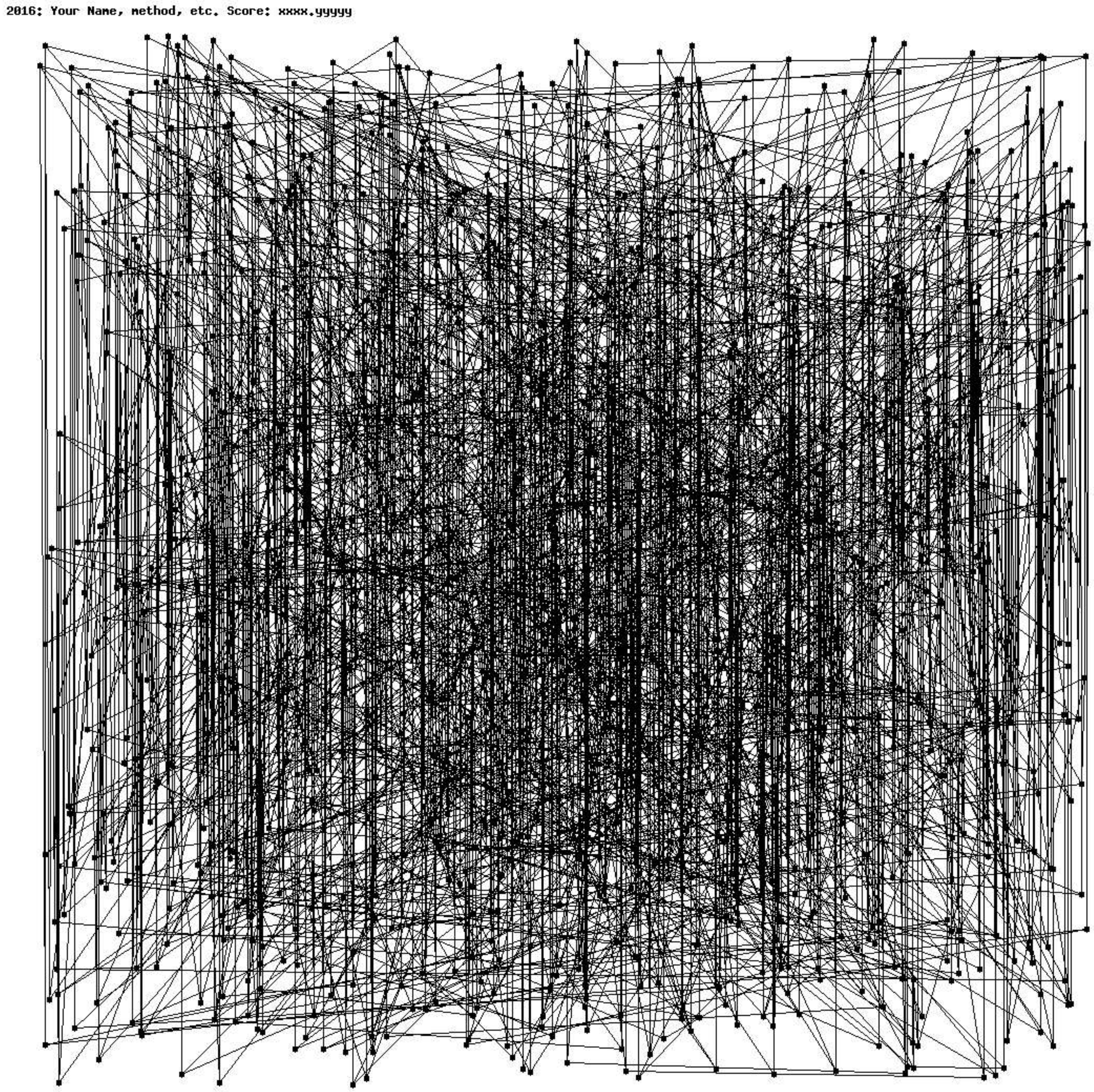
EX1: Below is the implementation of nearest neighbour algorithm.

In [1]:
```
import math, copy, sys, numpy, itertools

from itertools import permutations
from math import hypot
import numpy as np
from time import time

def parse(filename):
    with open(filename, "r") as ins:
        ins.readline()
        graph = []
        for line in ins:
            line = tuple(map(int, line.split(" ")))
            graph.append(line)
    return graph


def nn_algo(start_point, graph):
    g = copy.deepcopy(graph)
    path = [start_point]
    current_node = start_point
    while len(g) > 1:
        g.remove(current_node)
        distances = {}
        for i in g:
            distances[i] = math.sqrt((i[0] - current_node[0])**2 + (i[1] - cur
rent_node[1])**2)
        next_node = min(distances)
        current_node = next_node
        path.append(current_node)
    return path

def paint(startgraph, path, swogfile):
    swog = open(swogfile, "a")
    swog.write("\n\n")
    drawline = "line (p%d) (p%d)\n"
    for i in range(0, len(path)):
        if i == len(path) - 1 :
            swog.write(drawline % (startgraph.index(path[i]) + 1, startgraph.i
ndex(path[0]) + 1))
        else:
            swog.write(drawline % (startgraph.index(path[i]) + 1, startgraph.i
ndex(path[i + 1]) + 1))
    swog.close()



ex2 = [20,100,1000]
for i in ex2:
    graph = parse("2018/TSP_" + str(i) + ".txt")
    path = nn_algo(graph[0], graph)
    paint (graph, path, "2018/TSP_" + str(i) + ".swog")
```

The results are shown on the images below




2016: Your Name, method, etc. Score: xxxx.yyyyy

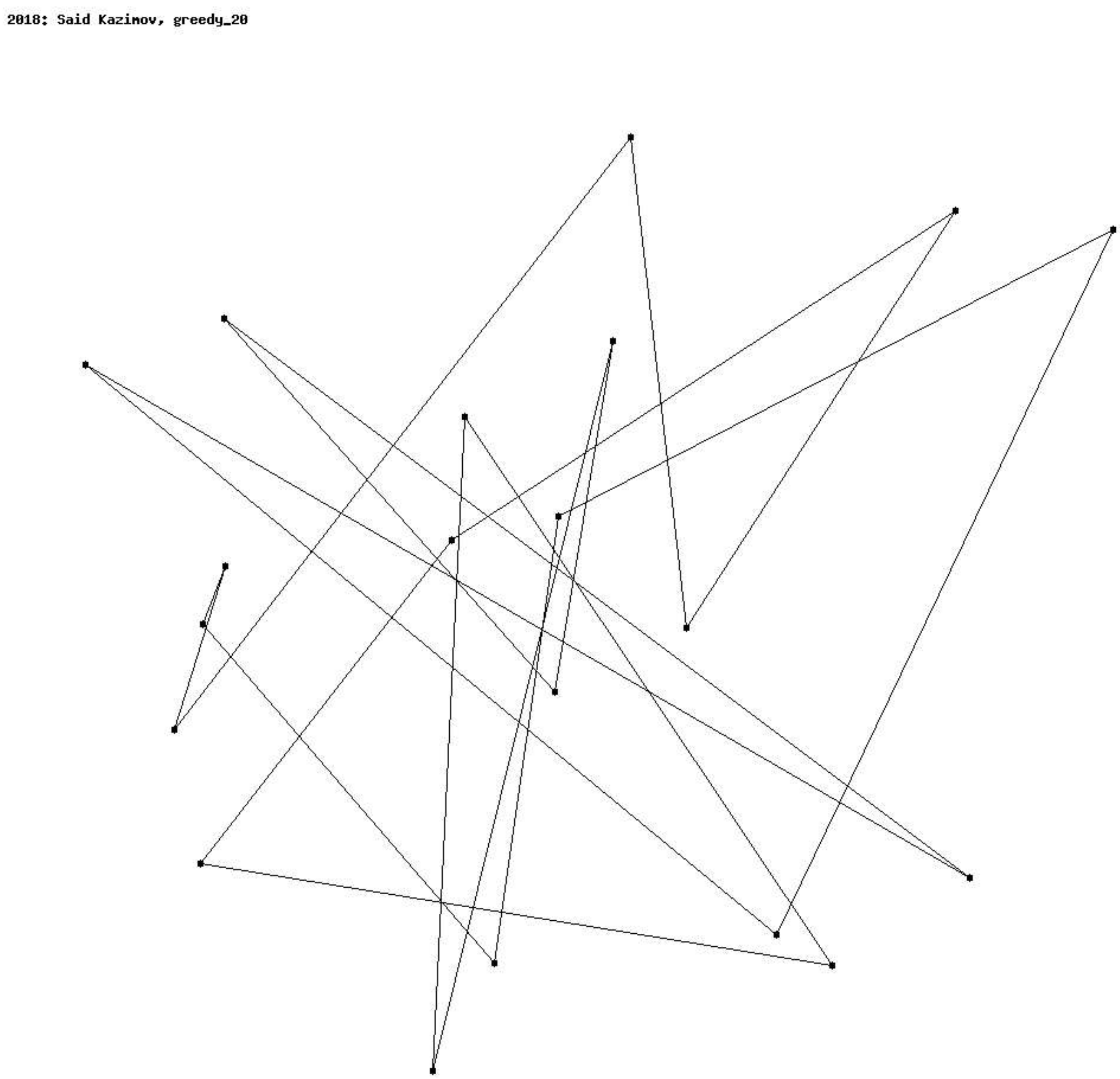


EX2&EX3:For optimization I chose greedy algorithm. Actually, it's Based on Kruskal's algorithm. It only gives a suboptimal solution in general. Works for complete graphs. May not work for a graph that is not complete. Below i showed the path that was found by this algorithm for 20 and 16 "cities"

In [2]:
```
def algorithm(cities):
    best_order = []
    best_length = float('inf')

    for i_start, start in enumerate(cities):
        order = [i_start]
        length = 0

        i_next, next, dist = get_closest(start, cities, order)
        length += dist
        order.append(i_next)

        while len(order) < cities.shape[0]:
            i_next, next, dist = get_closest(next, cities, order)
            length += dist
            order.append(i_next)


        if length < best_length:
            best_length = length
            best_order = order

    return best_order, best_length

def get_closest(city, cities, visited):
    best_distance = float('inf')

    for i, c in enumerate(cities):

        if i not in visited:
            distance = dist_squared(city, c)

            if distance < best_distance:
                closest_city = c
                i_closest_city = i
                best_distance = distance

    return i_closest_city, closest_city, best_distance

def dist_squared(c1, c2):
    t1 = c2[0] - c1[0]
    t2 = c2[1] - c1[1]

    return t1**2 + t2**2

f = open("2018/TSP_20.txt", 'r').read().splitlines()
numCities = f.pop(0)
cities = np.array([ tuple( map( int, coord.split() ) ) for coord in f ])
# print(cities)
#calculating path
start = time()
path, length = algorithm( cities )
print(length)

tottime = time() - start
print( "Found path of length %s in %s seconds" % (round(length,2),round(tottim
```

```
e, 2)))

graph = parse("2018/TSP_20.txt")
print(path)
path_nodes = []
for i in path:
    path_nodes.append(graph[i])
paint (graph, path_nodes, "2018/TSP_20.swog")
```

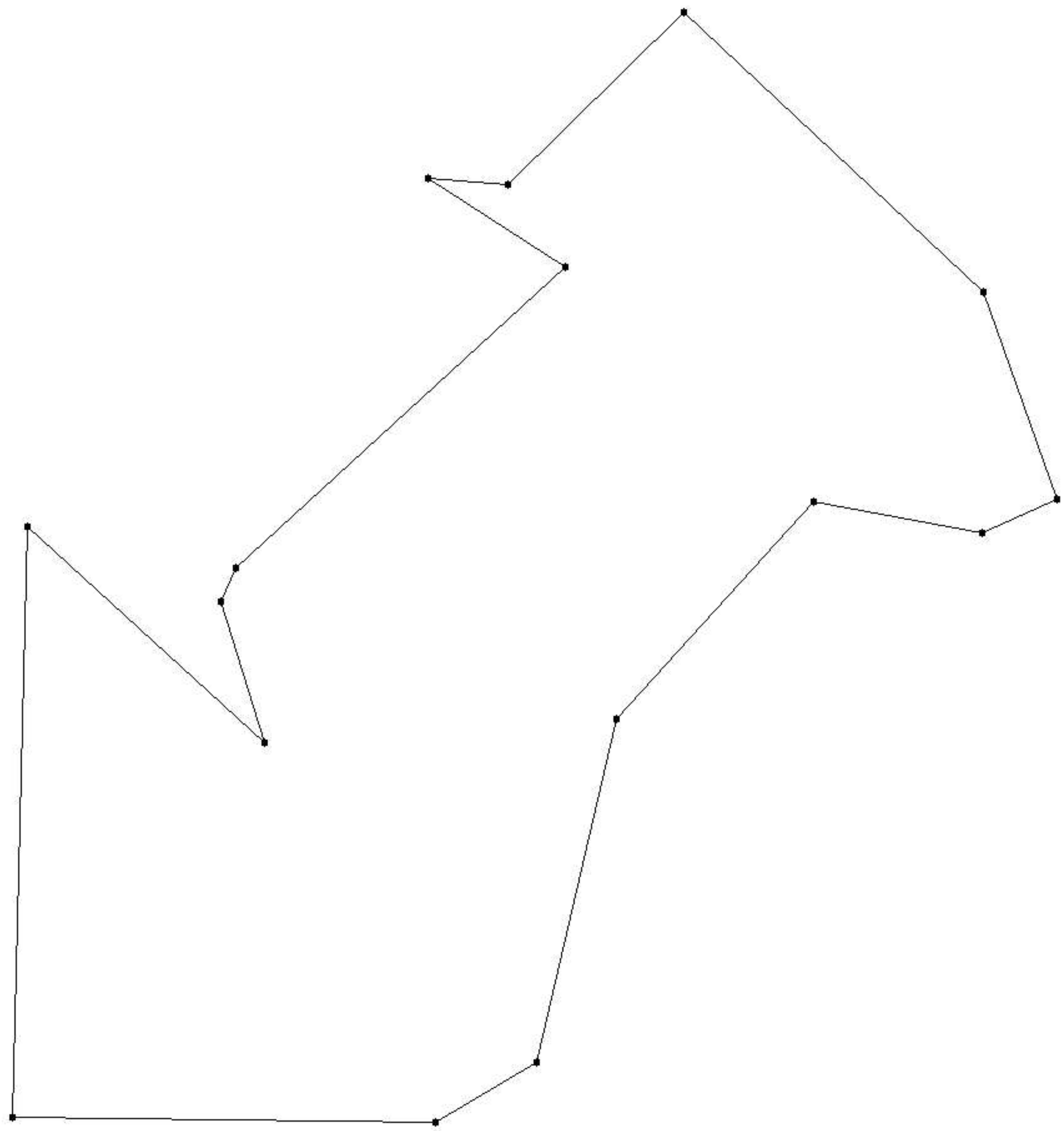
```
570647
Found path of length 570647 in 0.01 seconds
[14, 16, 10, 18, 11, 17, 1, 13, 5, 3, 12, 6, 9, 2, 8, 7, 0, 4, 15]
```



In [47]:
```
f = open("2018/TSP_16.txt", 'r').read().splitlines()
numCities = f.pop(0)
cities = np.array([ tuple( map( int, coord.split() ) ) for coord in f ])
start = time()
path, length = algorithm( cities )
print(length)

tottime = time() - start
print( "Found path of length %s in %s seconds" % (round(length,2),round(tottim
e, 2)))

graph = parse("2018/TSP_16.txt")
print(path)
path_nodes = []
for i in path:
    path_nodes.append(graph[i])
paint (graph, path_nodes, "2018/TSP_16.swog")
```

```
689615
Found path of length 689615 in 0.01 seconds
[15, 7, 6, 5, 14, 8, 10, 2, 12, 3, 11, 1, 4, 13, 0, 9]
```

2018: Said Kazimov, greedy_16



EX4&EX5:

1. Sudoku solving.
2. As we know there exist a few solving algorithms of sudoku, such as Backtracking, Stochastic search and etc. To understand how these algorithms work in our project we will try to visualize step-to-step change of this puzzle.
3. For those who are not familiar with the algorithms that I specified, we will provide visualization of it, so it would be more understandable, also, we will try to visually present which of the algorithms are faster(may be it can be depended on the various state of the cells).
4. Gantt chart:

In [10]:
```
from IPython.display import HTML, display

data = [["","19-31 December","1-10 January","11-19 January"],
         ["A","Analyzing one or two algorithms","Implementing the algorithm(s)
 in the visualization tool","Test & Integration"],
         ["B","Analyzing one or two algorithms","Implementing the algorithm(s)
 in the visualization tool","Test & Integration"],
         ["C","Application layer concept","Code application layer client/serve
r socket","Test & Integration"]]

display(HTML(
'<table><tr>{}</tr></table>'.format(
    '</tr><tr>'.join(
        '<td>{}</td>'.format('</td><td>'.join(str(_) for _ in row)) for row in
 data)
    )
))
```

	19-31 December	1-10 January	11-19 January
A	Analyzing one or two algorithms	Implementing the algorithm(s) in the visualization tool	Test & Integration
B	Analyzing one or two algorithms	Implementing the algorithm(s) in the visualization tool	Test & Integration
C	Application layer concept	Code application layer client/server socket	Test & Integration

1. Provide the results - which algorithm is actually faster and to show the demo of visualization tool on poster session

EX7: For me the most useful topics were dynamic programming and graphs, because before the course I did not have good understanding in these topics. In my opinion, the data structures should be covered more deeply and talking about, for example, 5 data structures in one lecture is not efficient. Topics that would need more practical implementation assignments are the exercises related to graphs. The first lectures about order of growth,linear structures and sorting seemed a bit boring for me, because it was the topics that I already know.