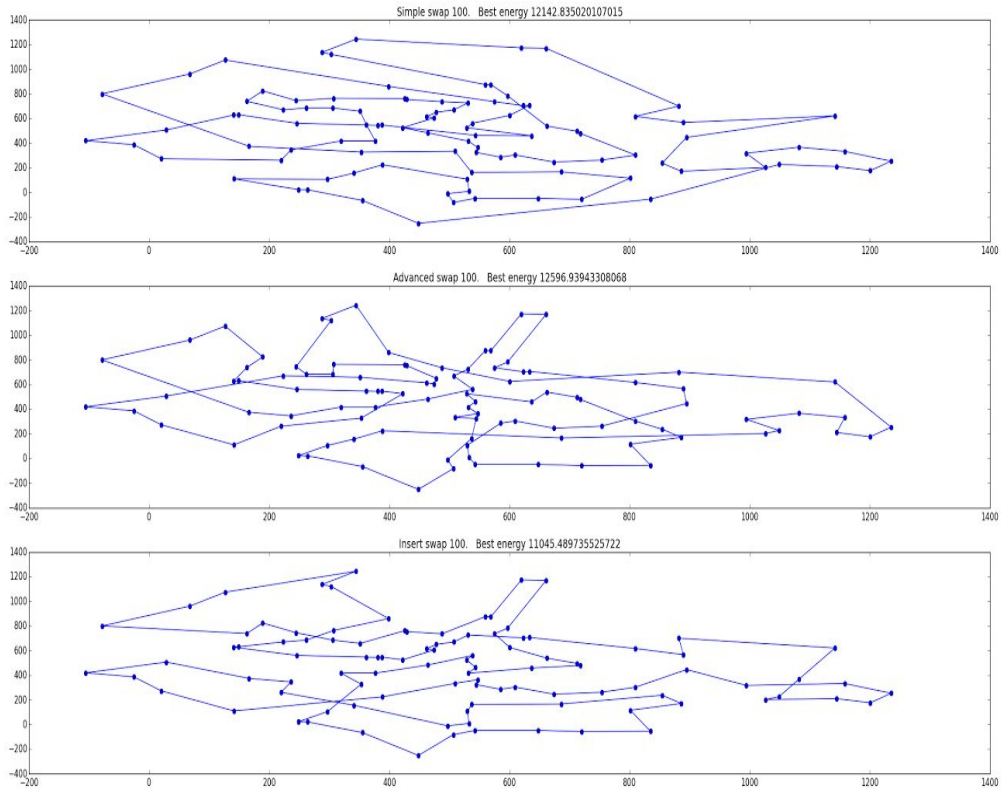
Jan Górski

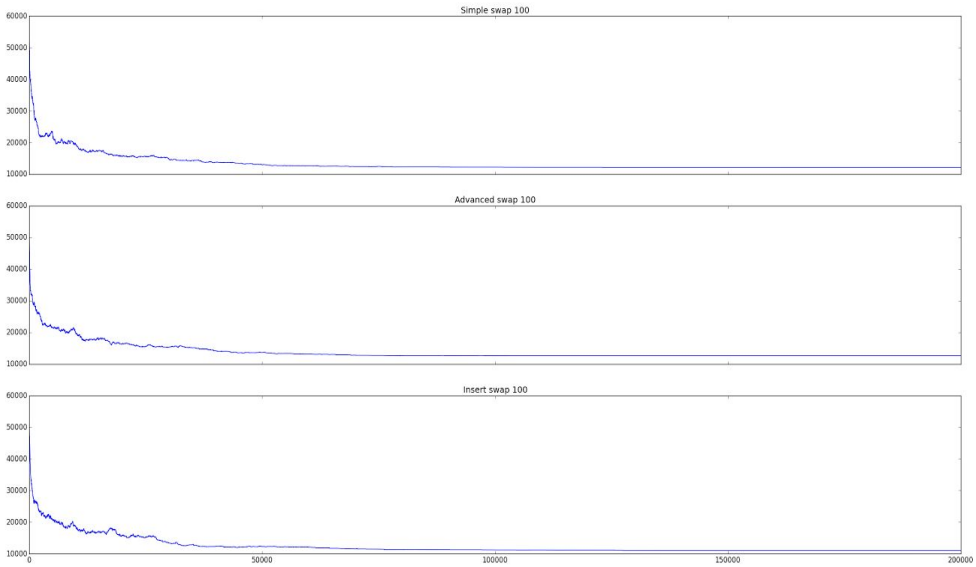
Sprawozdanie ćw 4

Zad 1

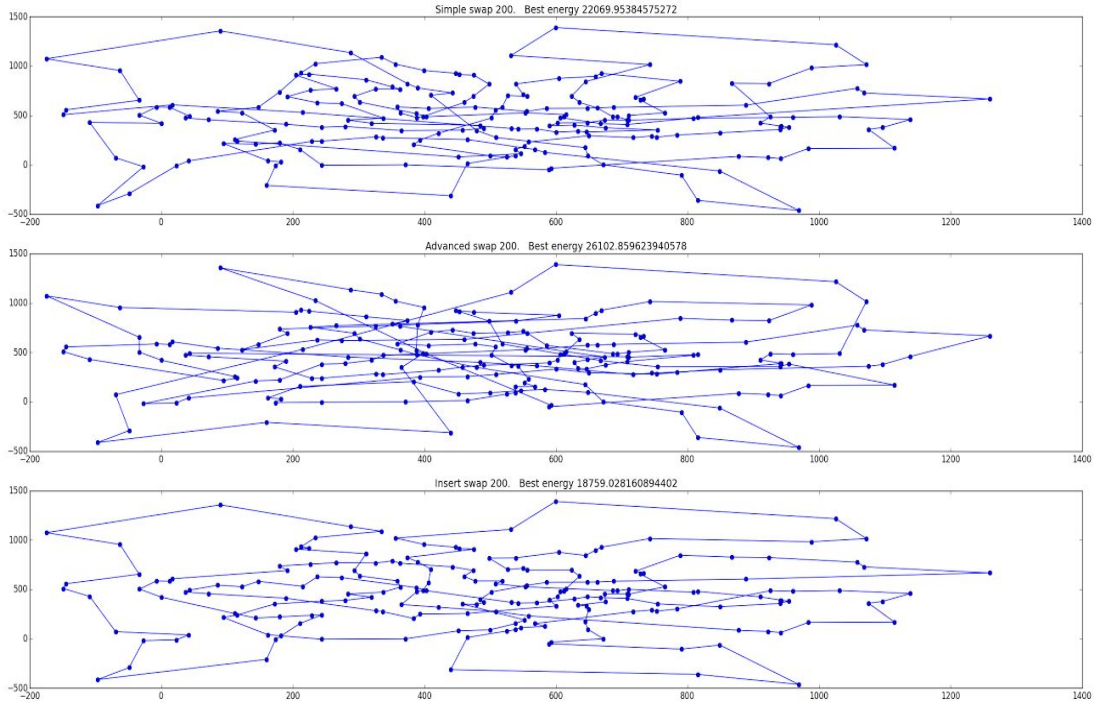
Liczba punktów n = 100, generacja = rozkład naturalny



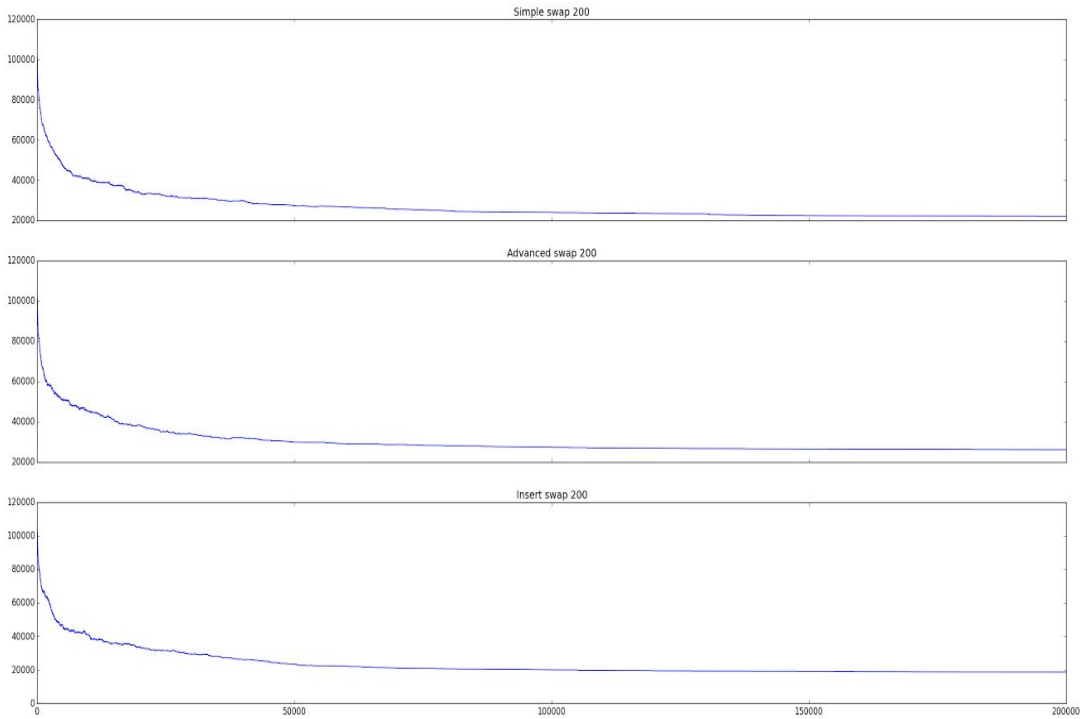
Energia:



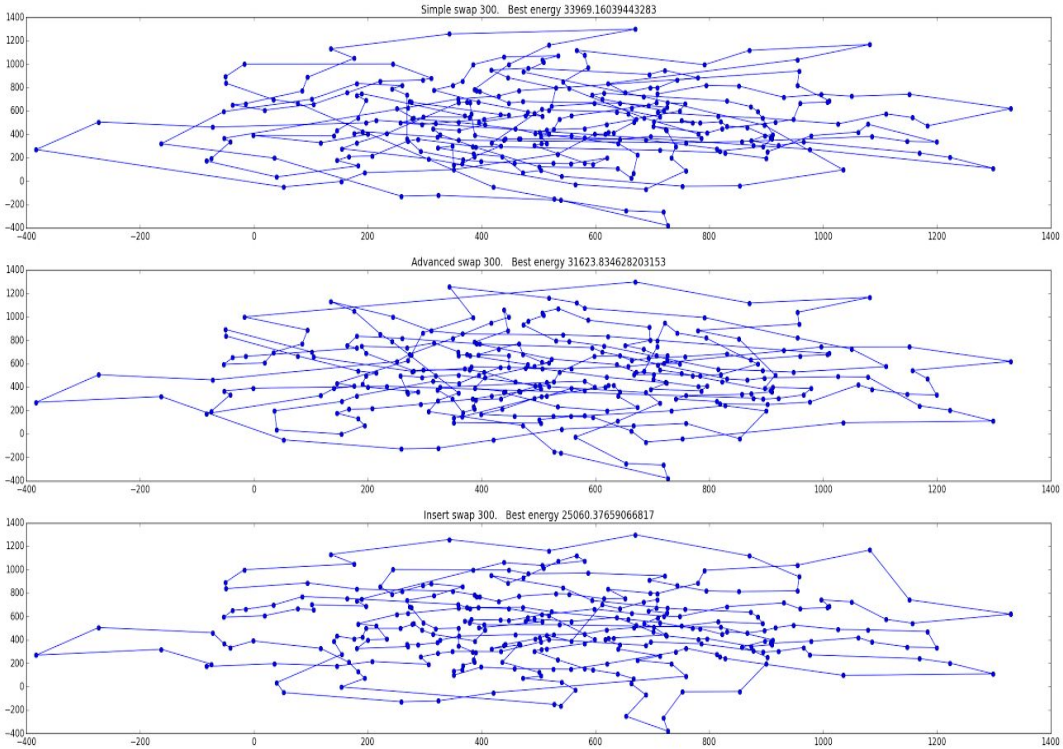
Liczba punktów n = 200, generacja = rozkład naturalny



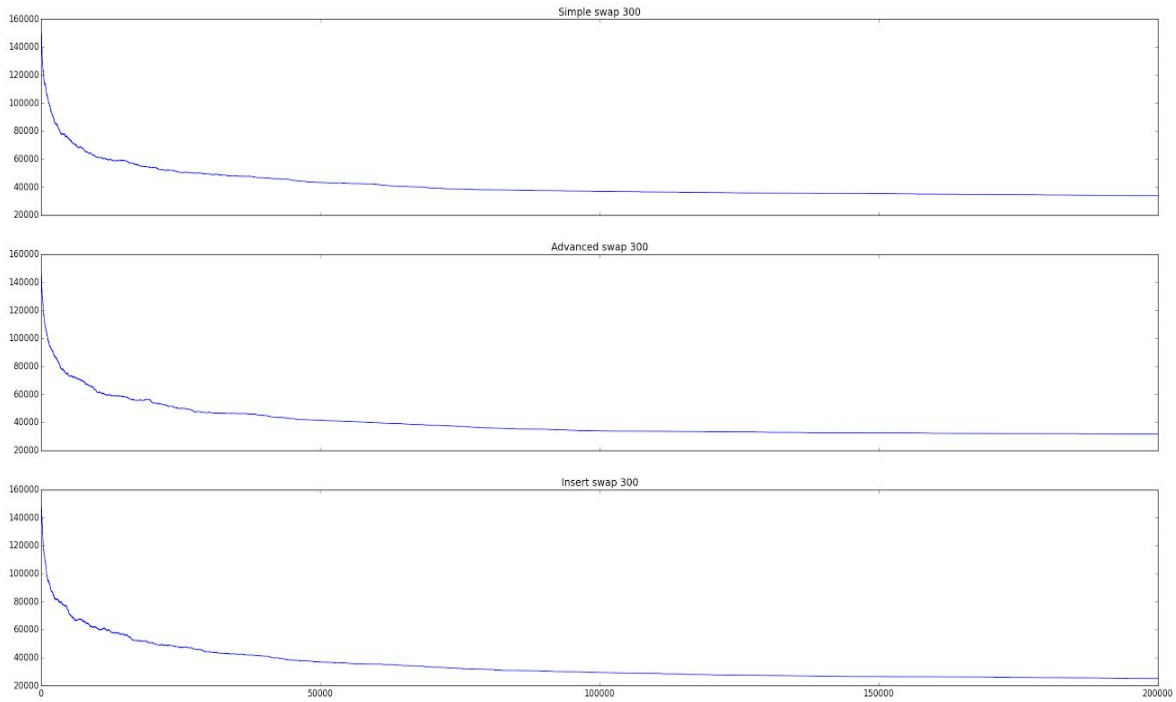
Energia:



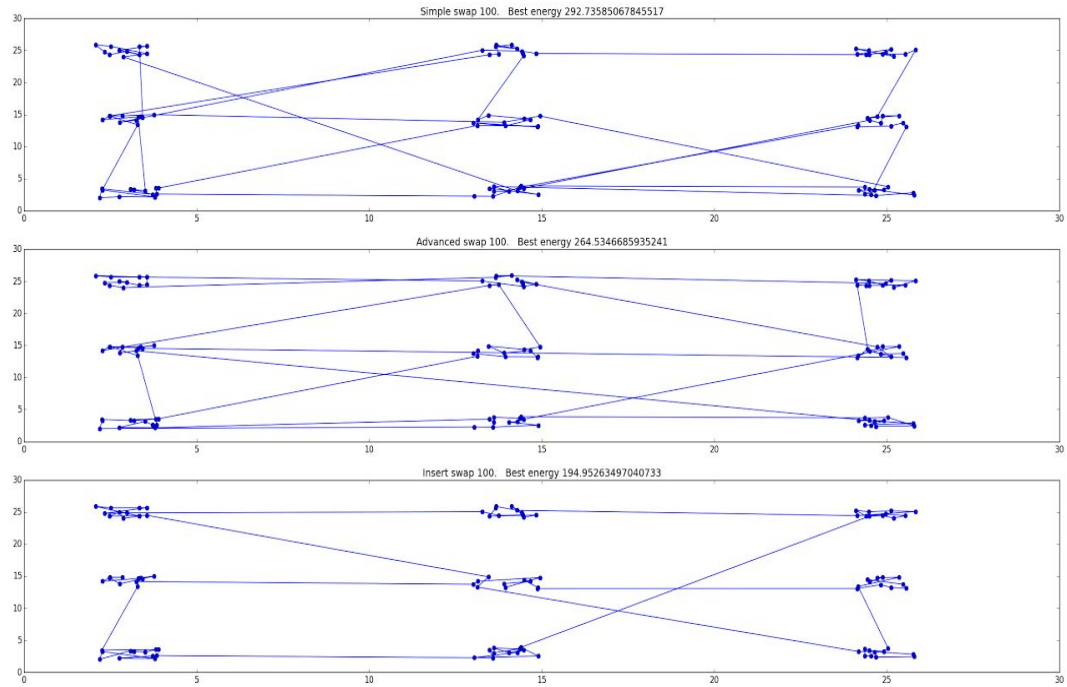
Liczba punktów n = 300, generacja = rozkład naturalny



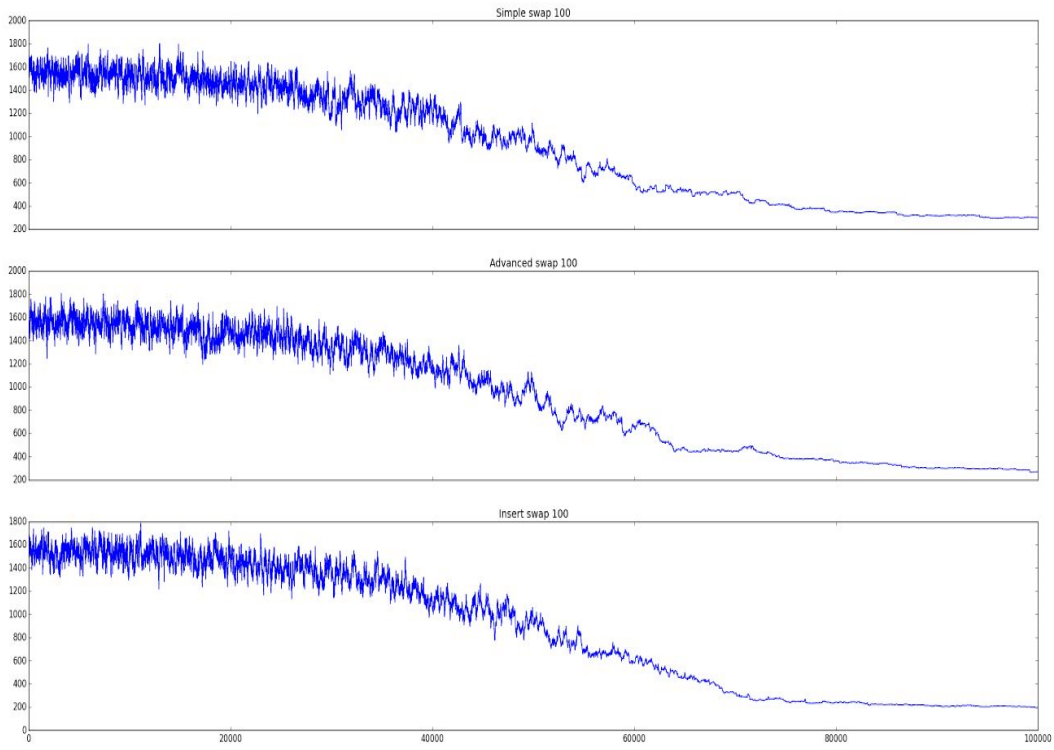
Energia:



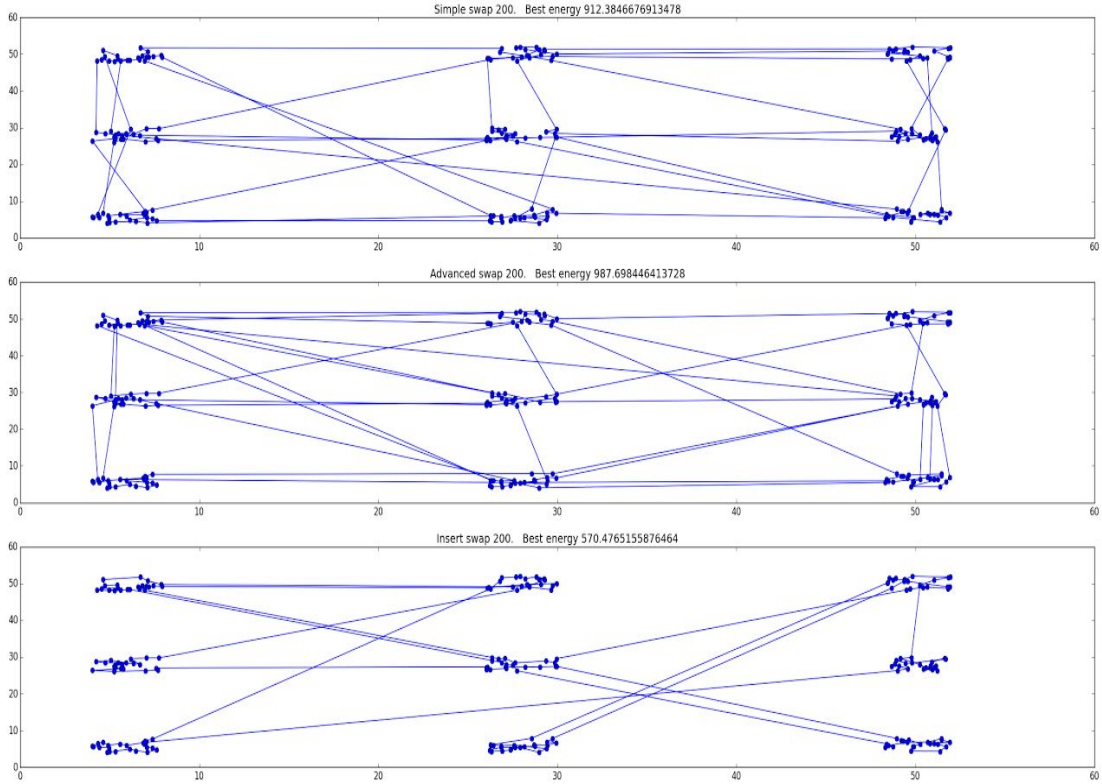
Liczba punktów n = 100, generacja = 9 oddzielnych grup



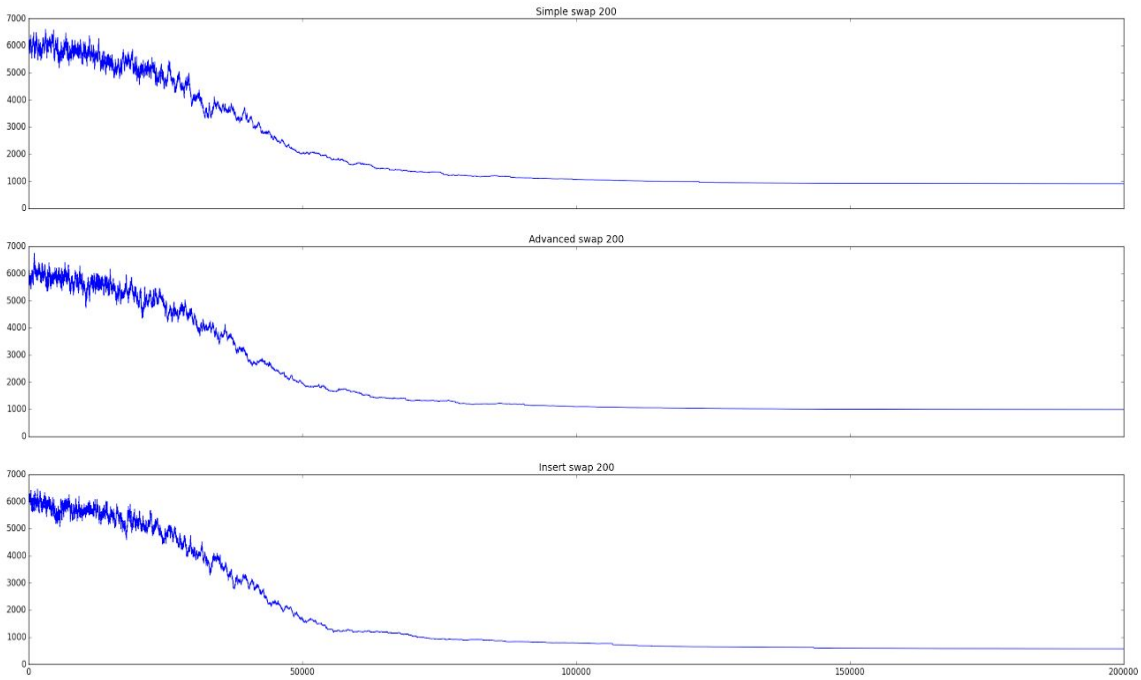
Energia:



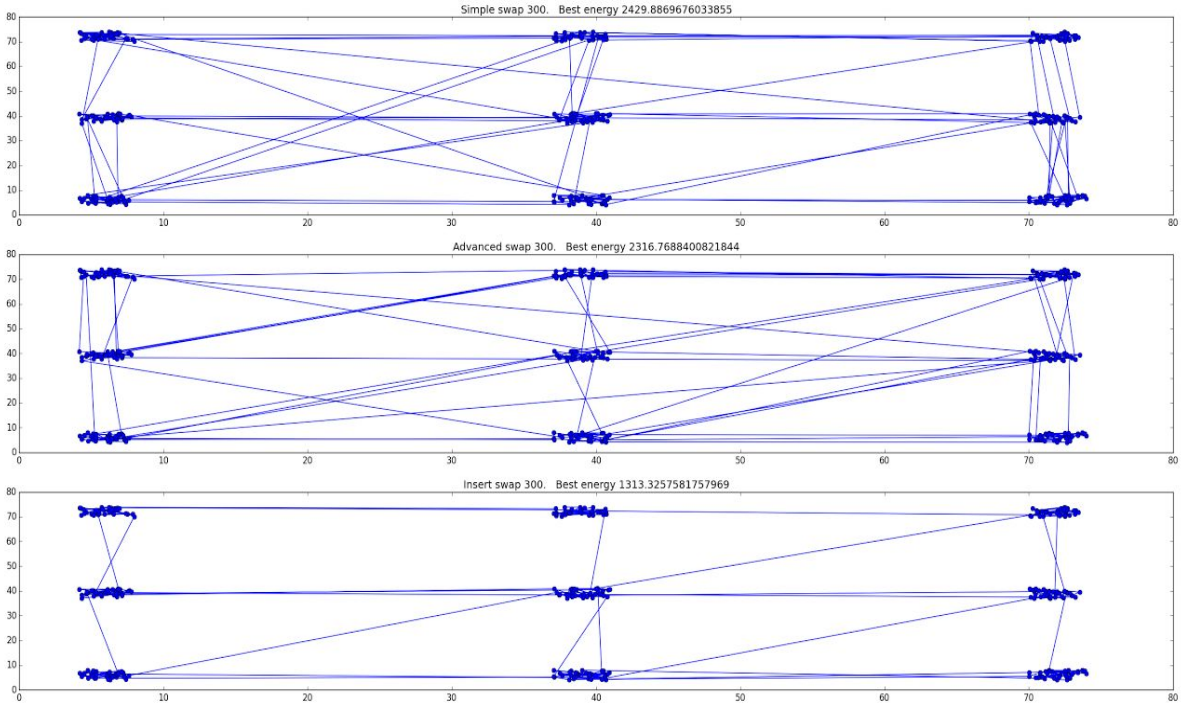
Liczba punktów n = 200, generacja = 9 oddzielnych grup



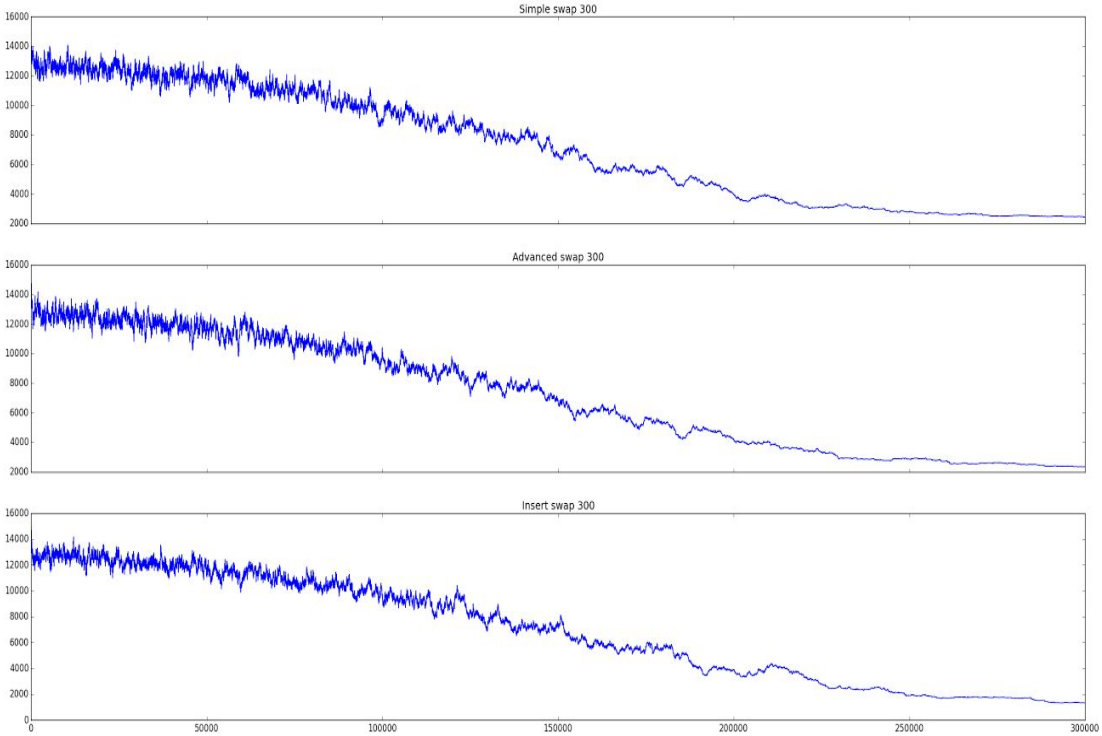
Energia:



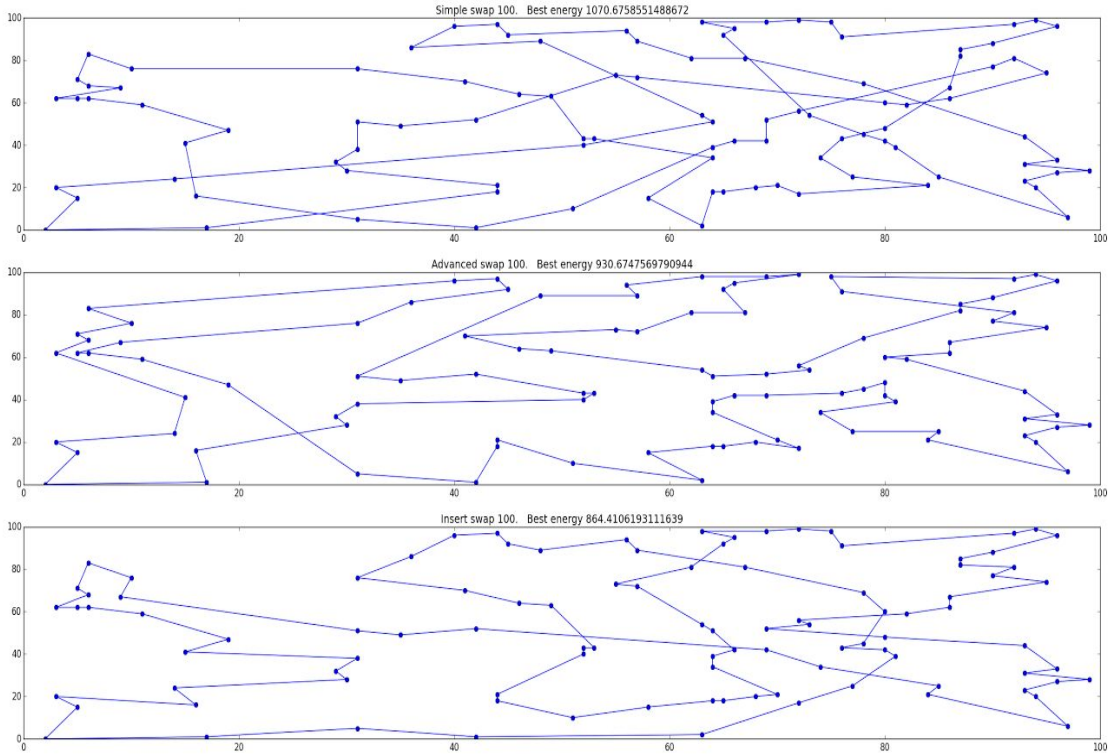
Liczba punktów n = 300, generacja = 9 oddzielnych grup



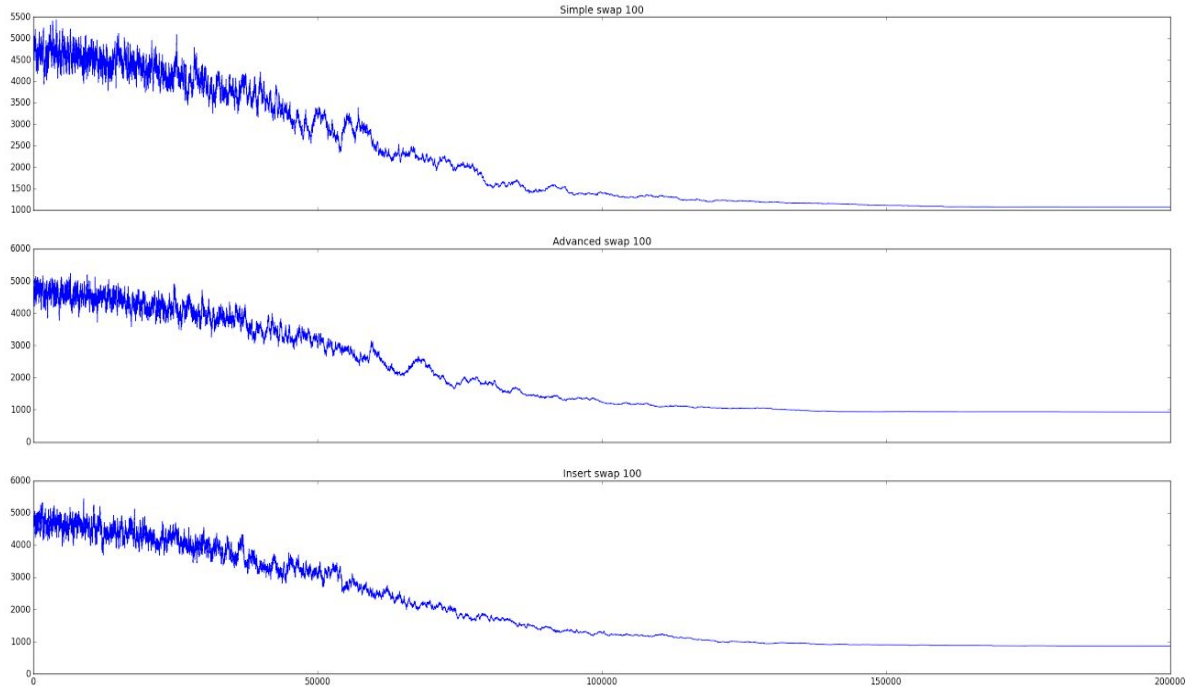
Energia:



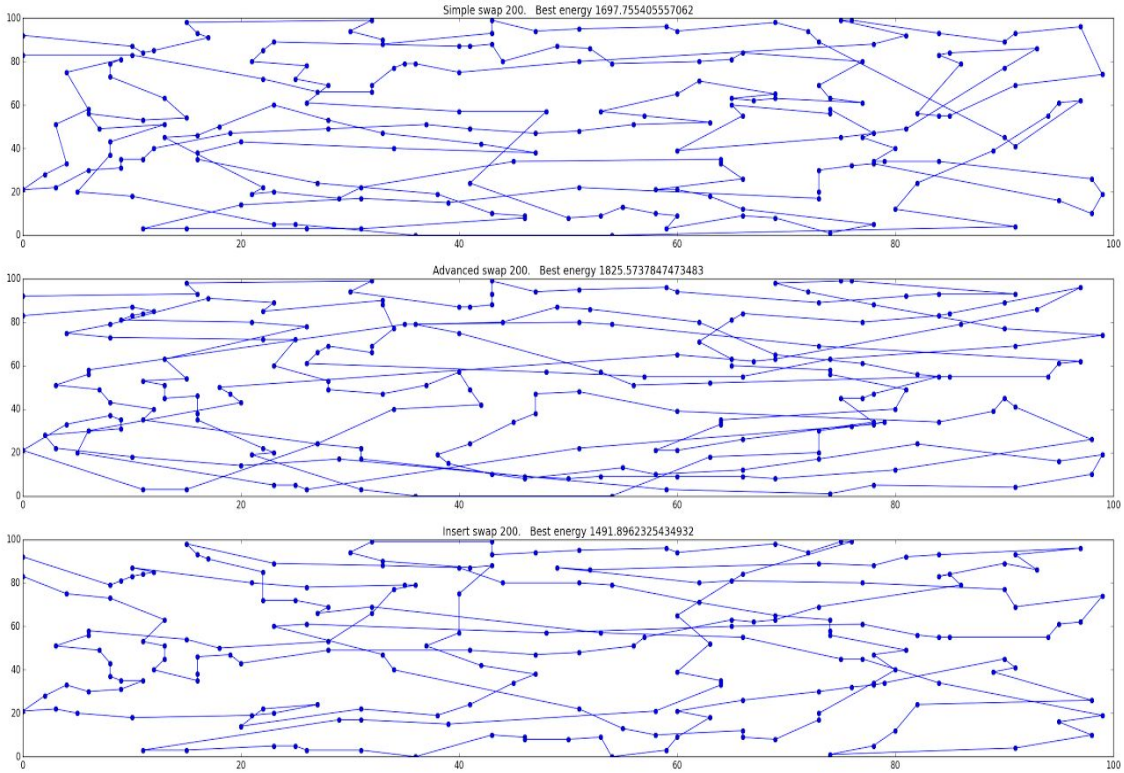
Liczba punktów n = 100, generacja = losowy jednostajny



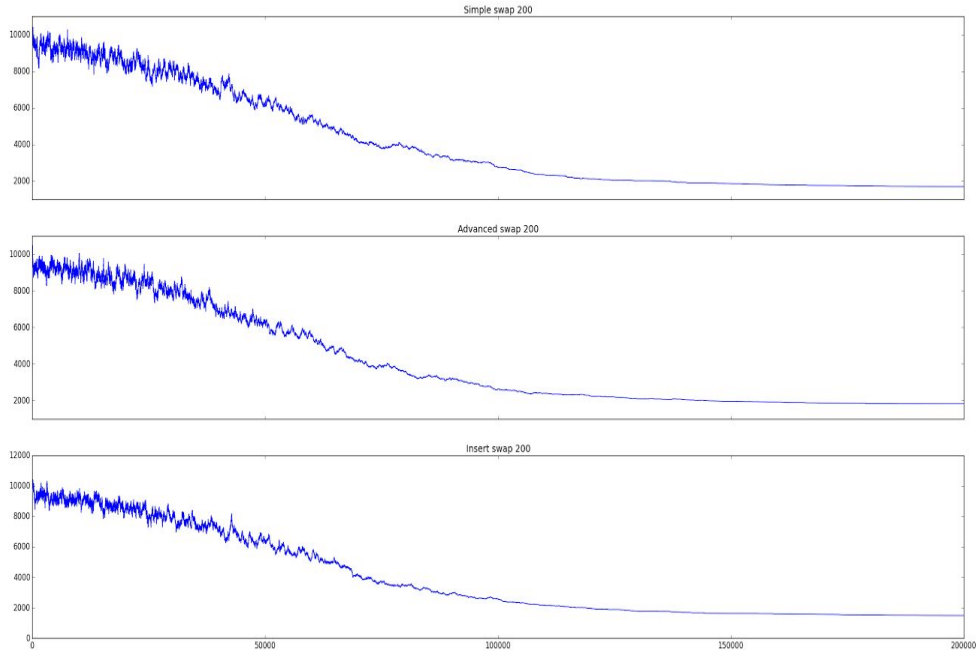
Energia;



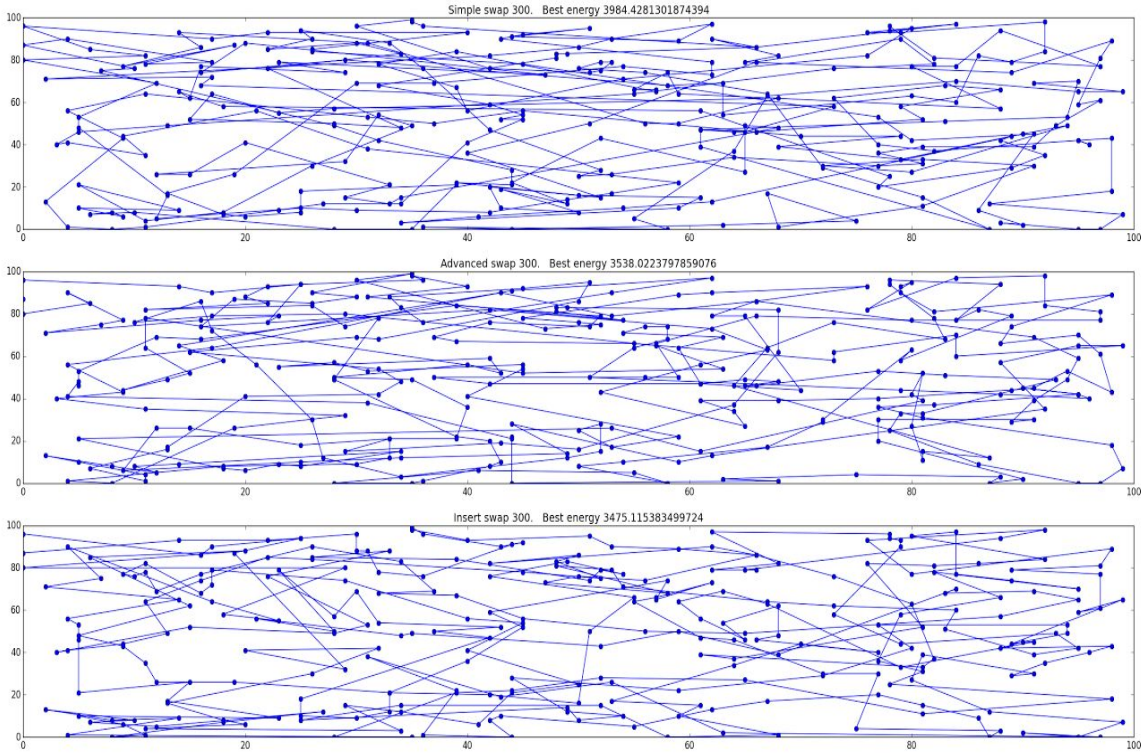
Liczba punktów n = 200, generacja = losowy jednostajny



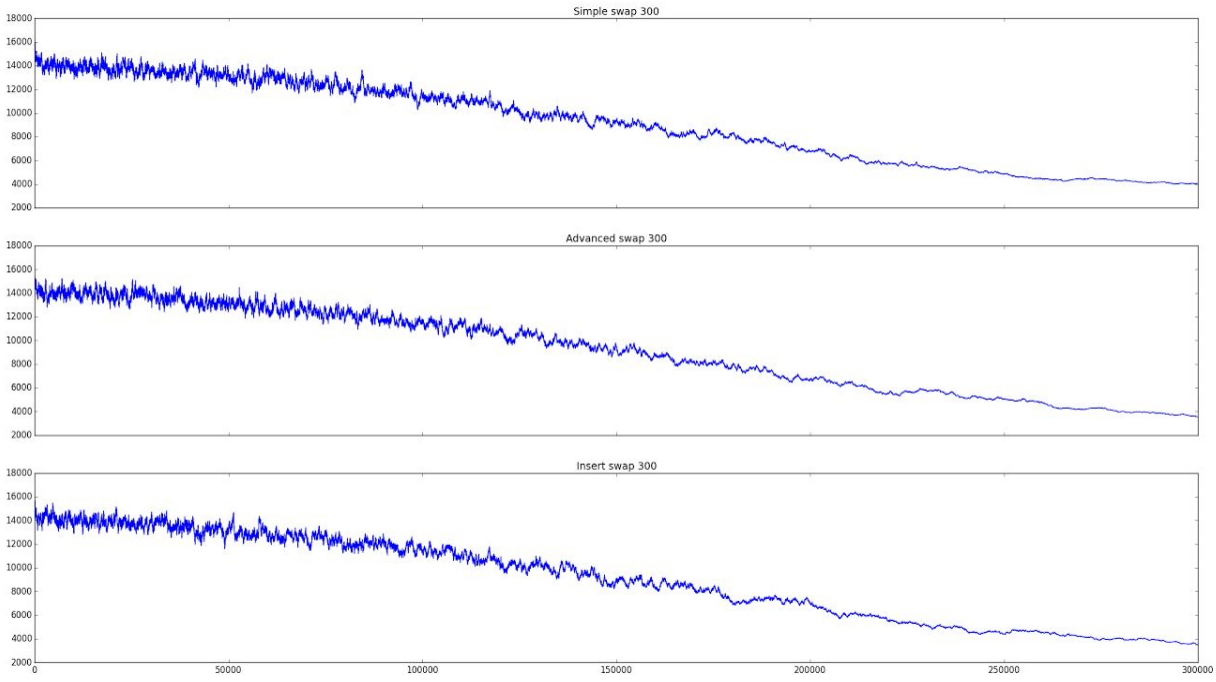
Energia:



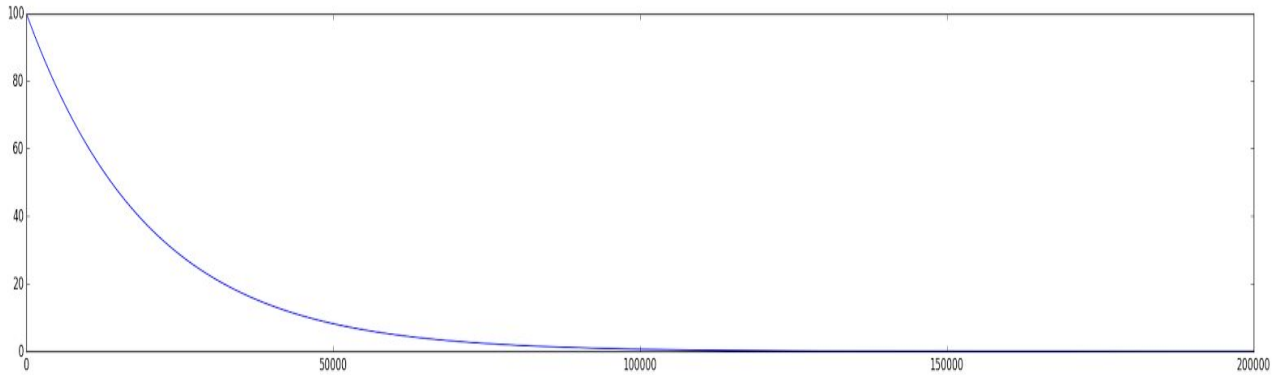
Liczba punktów n = 300, generacja = losowy jednostajny



Energia:



Wykres temperatury dla powyższych wykresów:



KOD:

```
import copy
import math
import random
import numpy as np
import matplotlib.pyplot as plt


def temp(T, n):
    return T * n


def distance(x1, y1, x2, y2):
    return math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)


def calculate(arr, n):
    energy = 0.0
    for i in range(n):
        energy += distance(arr[i][0], arr[i][1], arr[(i + 1) %
n][0], arr[(i + 1) % n][1])
    return energy


def simple_swap(arr, n ,k):
    new = [d[:] for d in arr]
    first = random.randrange(n)
    second = random.randrange(n)
    while first == second:
        second = random.randrange(n)
```

```
    new[first], new[second] = new[second], new[first]
    return new


def advanced_swap(arr, n, k):
    new = [d[:] for d in arr]
    first = random.randrange(n)
    new[first], new[(first + k) % (n-1) + 1] = new[(first + k) %
(n-1) + 1], new[first]
    return new


def insert_swap(arr, n, k):
    new = [d[:] for d in arr]
    first = random.randrange(n)
    second = random.randrange(n-1)
    while first == second:
        second = random.randrange(n-1)
    x = new.pop(first)
    new.insert(second,x)
    return new

def gen_four(n):
    sigma = 300
    mu = 500

    arr = list(sigma * np.random.randn(n, 2) + mu)
    return arr


def gen_nine(n):
    dx = n // 9
    b = n % 9
    z = random.randint(2,5)
    arr = []
    for a in range(dx):
        for y in range (3):
            for x in range(3):

arr.append([random.uniform(x*dx+z,x*dx+2*z),random.uniform(y*dx+z,
y*dx+2*z)])
                if b > 0:

arr.append([random.uniform(x*dx+z,x*dx+2*z),random.uniform(y*dx+z,
y*dx+2*z)])
                    b -= 1
```

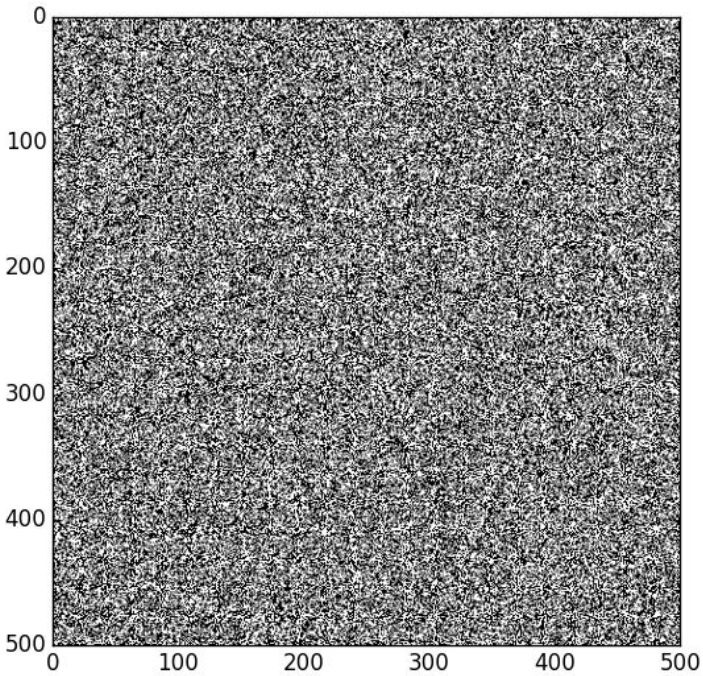
```
    random.shuffle(arr)
    return arr


def sim_annealing(swaping, arr, n, T, k):
    energy_val = []
    tem = []
    current_solution = copy.deepcopy(arr)
    current_val = calculate(current_solution, n)
    best_solution = current_solution
    best_val = current_val
    print("Beggining val: " + str(best_val) +" len: " +
str(len(arr)))
    for i in range(k):
        neighbour = swaping(current_solution, n, i)
        second_val = calculate(neighbour, n)
        if second_val < current_val:
            current_solution = neighbour
            current_val = second_val
        else:
            if math.exp((current_val - second_val) / T) >
random.random():
                current_solution = neighbour
                current_val = second_val
        if best_val > current_val:
            best_solution = current_solution
            best_val = current_val
        tem.append(T)
        T = temp(T, 0.99997)
        energy_val.append(current_val)
        if i % 1000 == 0:
            print("Calculated val: " + str(current_val) +"\tBest
val: " + str(best_val) + "\t T is :" + str(T))
    print("Calculated val: " + str(best_val))
    return best_solution, energy_val, tem, best_val


def main():
    T = 100.0
    k = 200000
    n = 300
    #arr = [[random.randrange(100) for i in range(2)] for x in
range(n)]
    arr = gen_four(n)
    f, (ax1, ax2, ax3) = plt.subplots(3)
    f, (bx1, bx2, bx3) = plt.subplots(3,sharex=True)
```

```
    f, (cx1, cx2, cx3) = plt.subplots(3,sharex=True)

    solution, energy, tem, best_val = sim_annealing(simple_swap,
arr, n, T, k)
    bx1.plot(range(len(energy)),energy)
    cx1.set_title('Simple swap '+str(n))
    cx1.plot(range(len(tem)),tem)
    bx1.set_title('Simple swap '+str(n))
    ax1.set_title('Simple swap '+str(n)+ ".   Best energy " +
str(best_val))
    ax1.plot([row[0] for row in solution] + [solution[0][0]],
[row[1] for row in solution] + [solution[0][1]],
             marker="o")

    solution, energy, tem, best_val = sim_annealing(advanced_swap,
arr, n, T, k)
    cx2.plot(range(len(tem)),tem)
    cx2.set_title('Advanced swap '+str(n))
    bx2.plot(range(len(energy)),energy)
    bx2.set_title('Advanced swap '+str(n))
    ax2.set_title('Advanced swap '+str(n)+ ".   Best energy " +
str(best_val))
    ax2.plot([row[0] for row in solution] + [solution[0][0]],
[row[1] for row in solution] + [solution[0][1]],
             marker="o")

    solution, energy, tem, best_val = sim_annealing(insert_swap,
arr, n, T, k)
    bx3.plot(range(len(energy)),energy)
    bx3.set_title('Insert swap ' +str(n))
    cx3.plot(range(len(tem)),tem)
    cx3.set_title('Insert swap ' +str(n))
    ax3.set_title('Insert swap ' +str(n)+ ".   Best energy " +
str(best_val))
    ax3.plot([row[0] for row in solution] + [solution[0][0]],
[row[1] for row in solution] + [solution[0][1]],
             marker="o")

    plt.show()


if (__name__ == "__main__"):
    main()
```

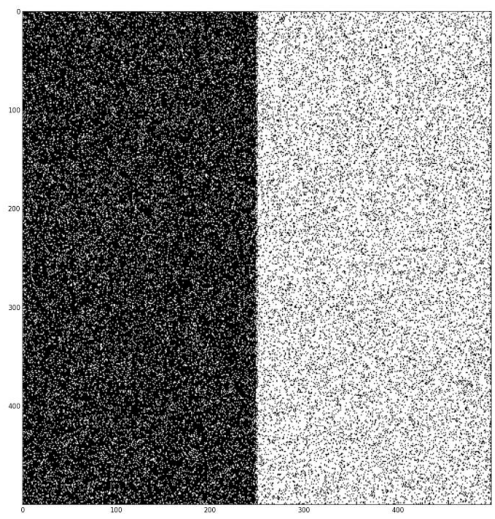
Zad 2
Parametry dla wszystkich obrazów
Temperatura początkowa = 100.0

Obrazy:
500x500, density =0.5, funkcja energii=energy1, funkcja sąsiedztwa=neighbour1, k =600000

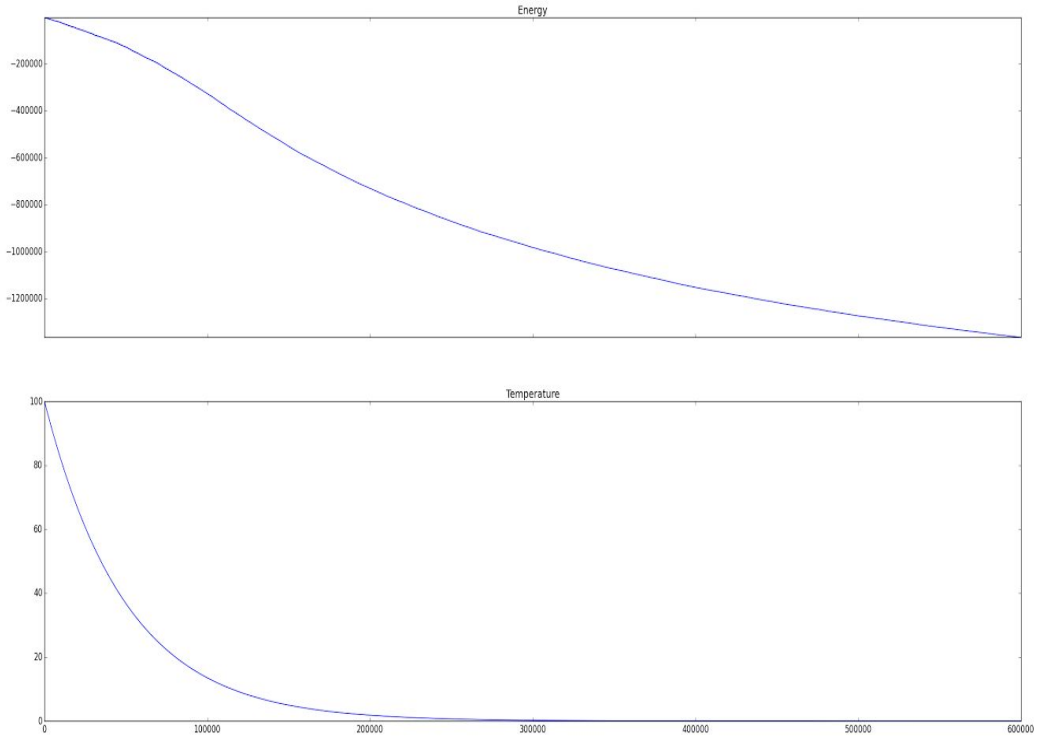
Przed wyżarzaniem:



Po wyżarzaniu:

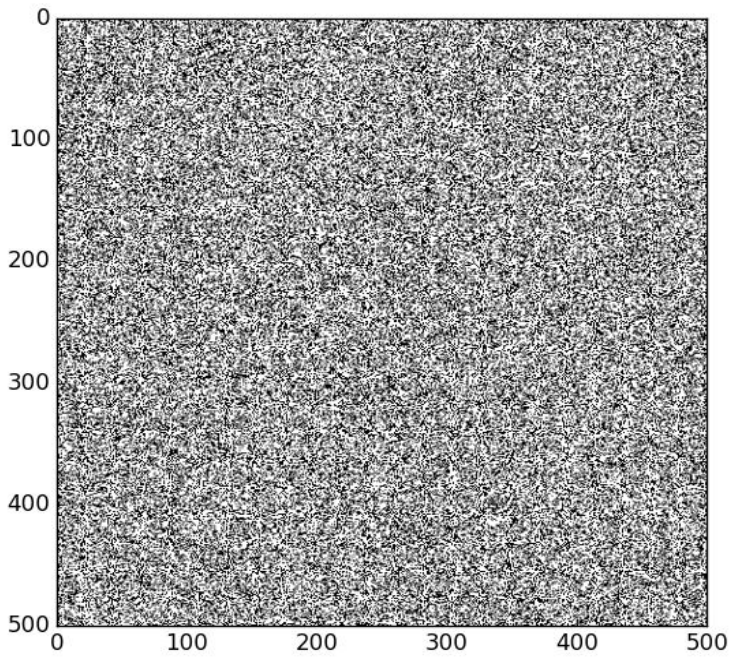


Energia i Temperatura:

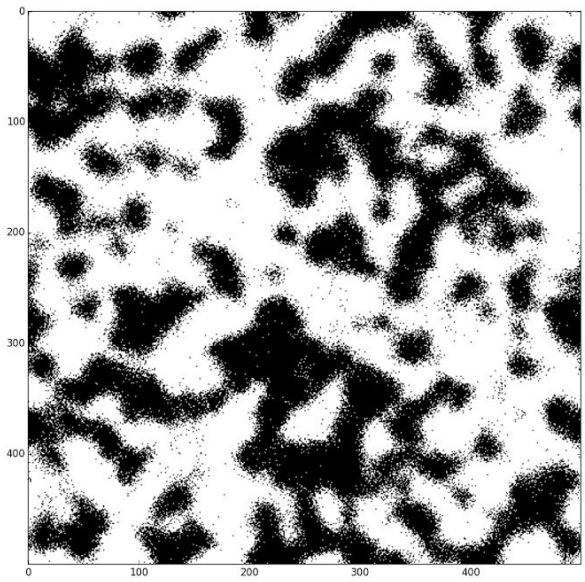


500x500, density=0.3, funkcja energii=energy3, funkcja sąsiedztwa=neighbour3, k=1000000

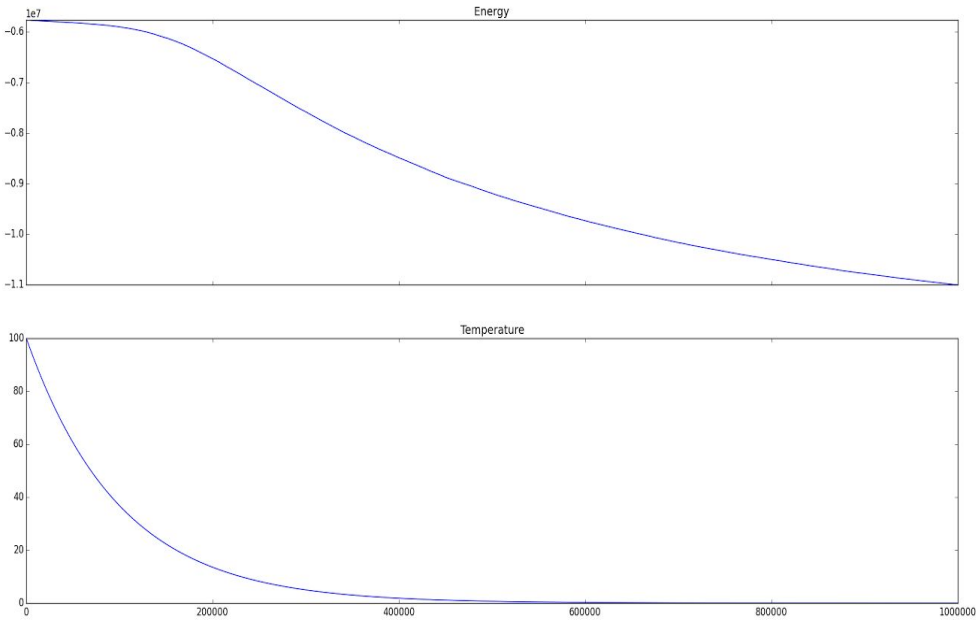
Przed wyżarzaniem:



Po wyżarzaniu:

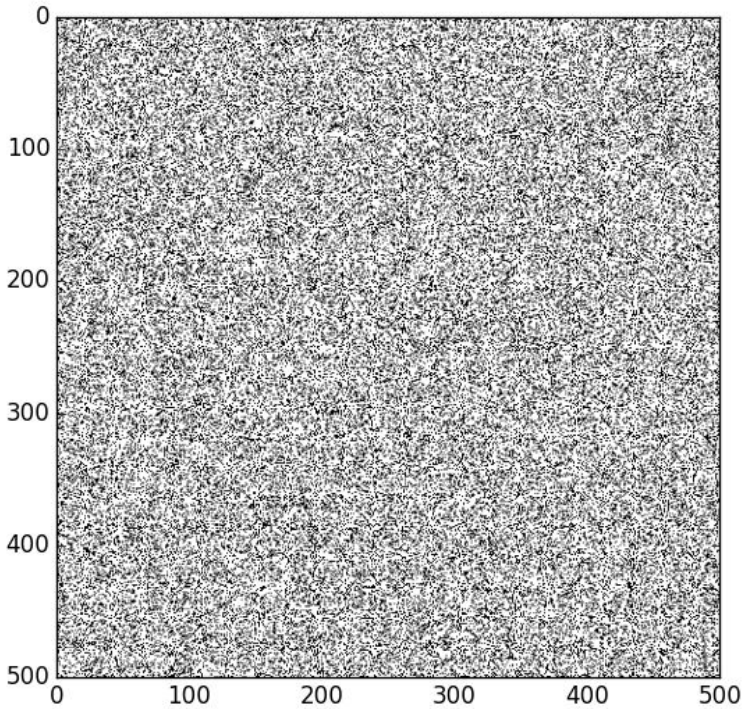


Energia i Temperatura:

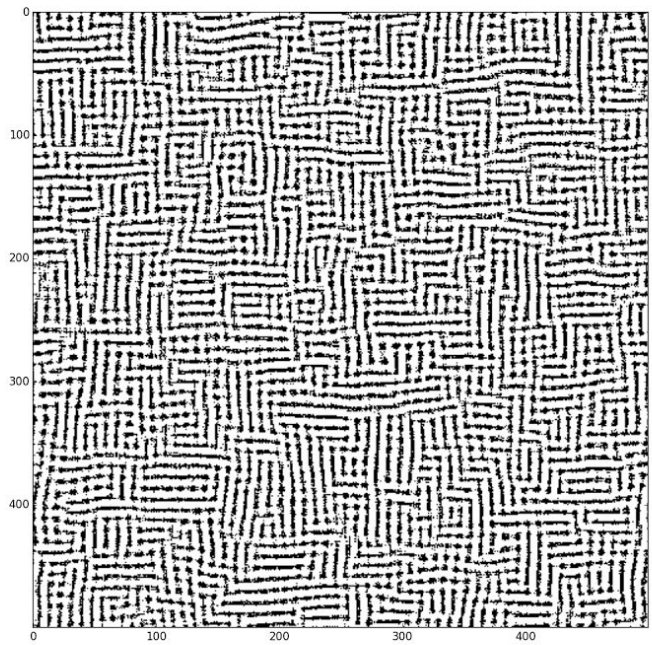


500x500, density=0.1, funkcja energii=energy4, funkcja sąsiedztwa=neighbour3, k=1000000

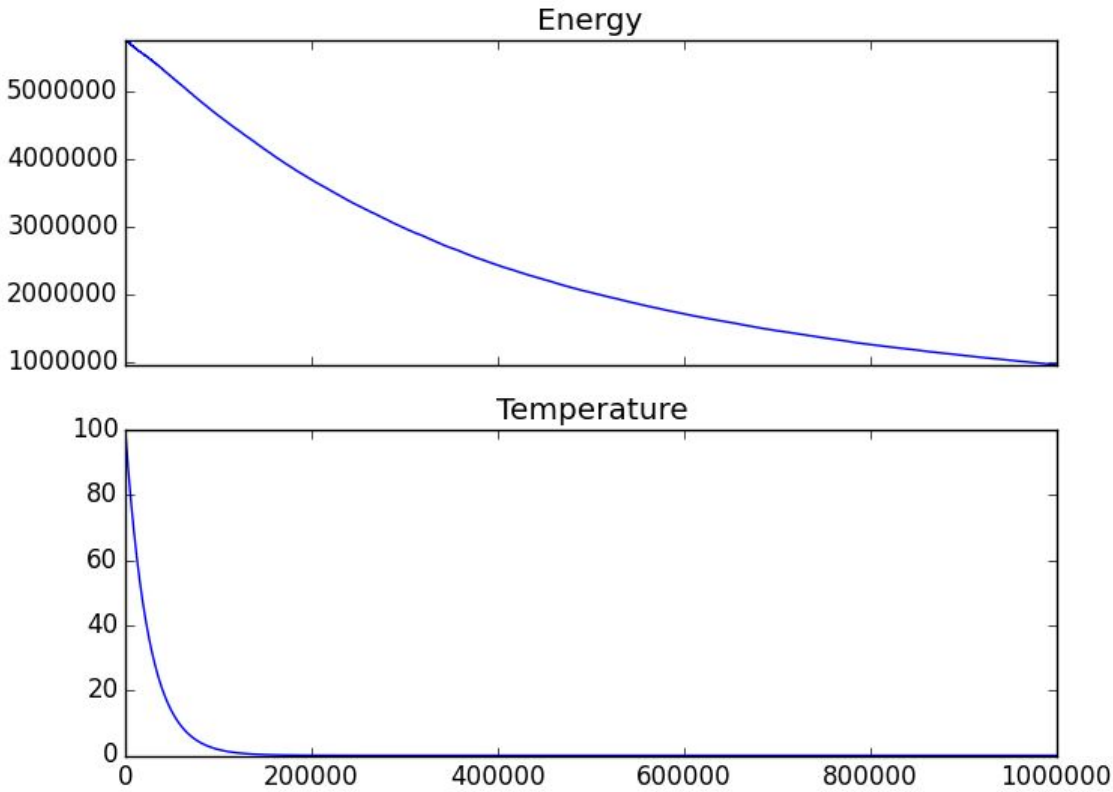
Przed wyżarzaniem:



Po wyżarzaniu:

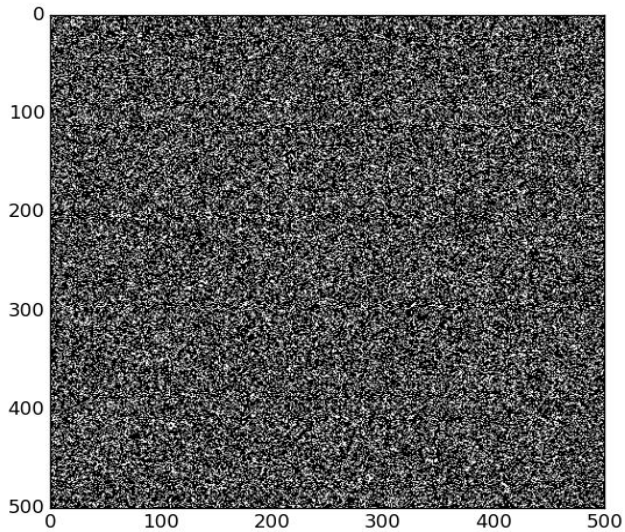


Energia i Temperatura:

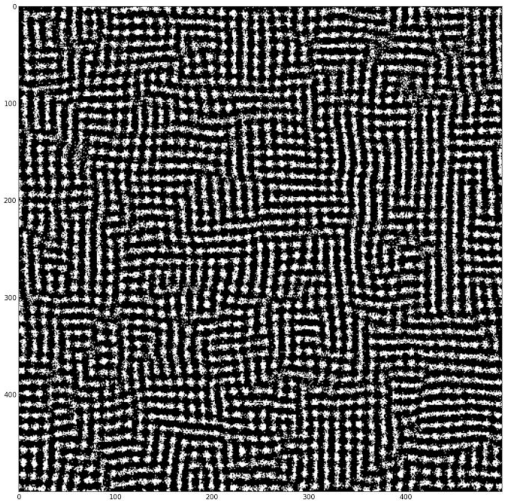


500x500, density=0.7, funkcja energii=energy4, funkcja sąsiedztwa=neighbour3, k=600000

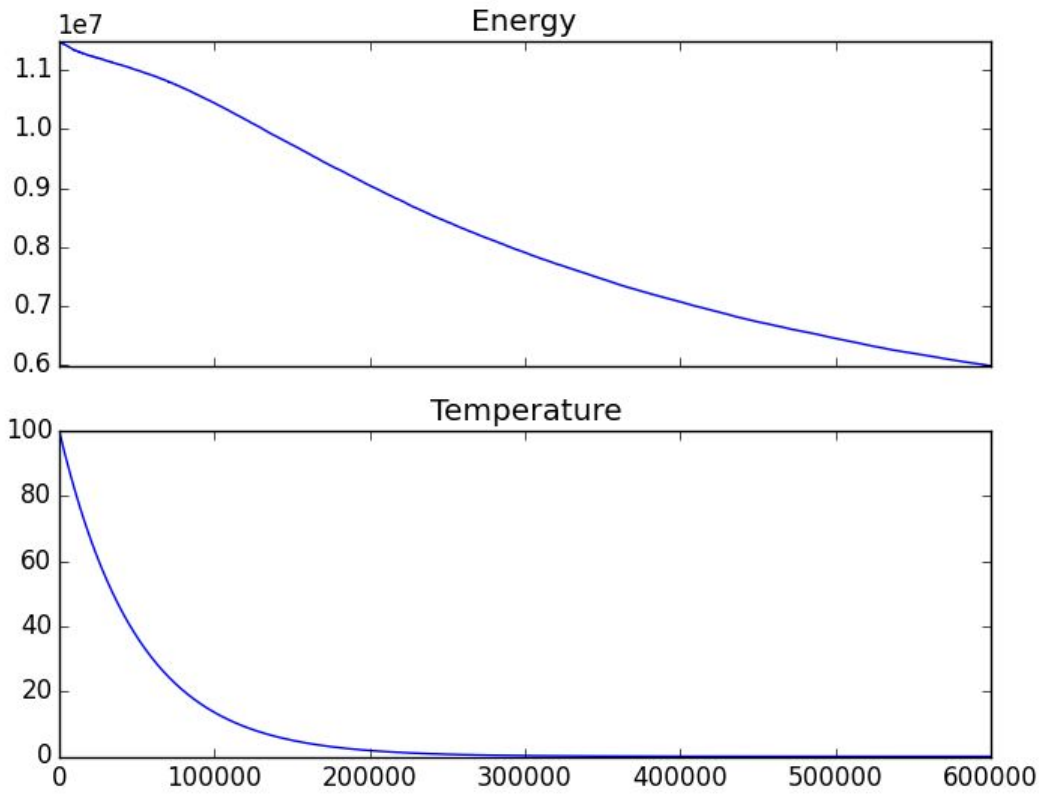
Przed wyżarzaniem:



Po wyżarzaniu:

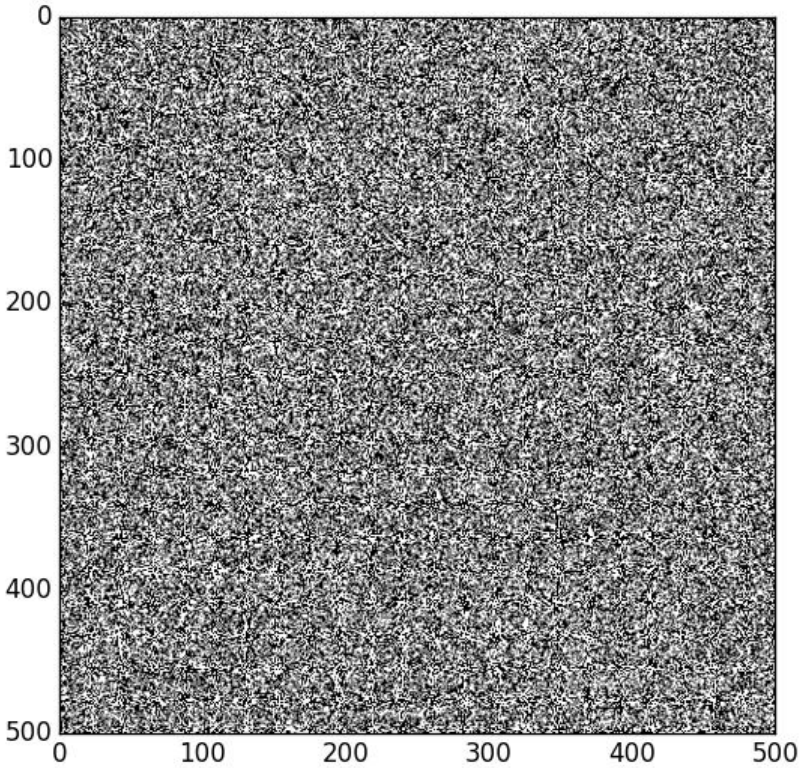


Energia i Temeratura:

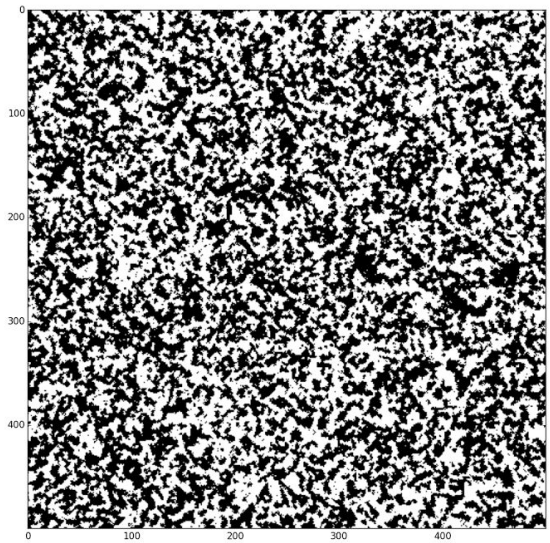


500x500, density=0.5, funkcja energii=energy5, funkcja sąsiedztwa=neighbour2, k=1000000

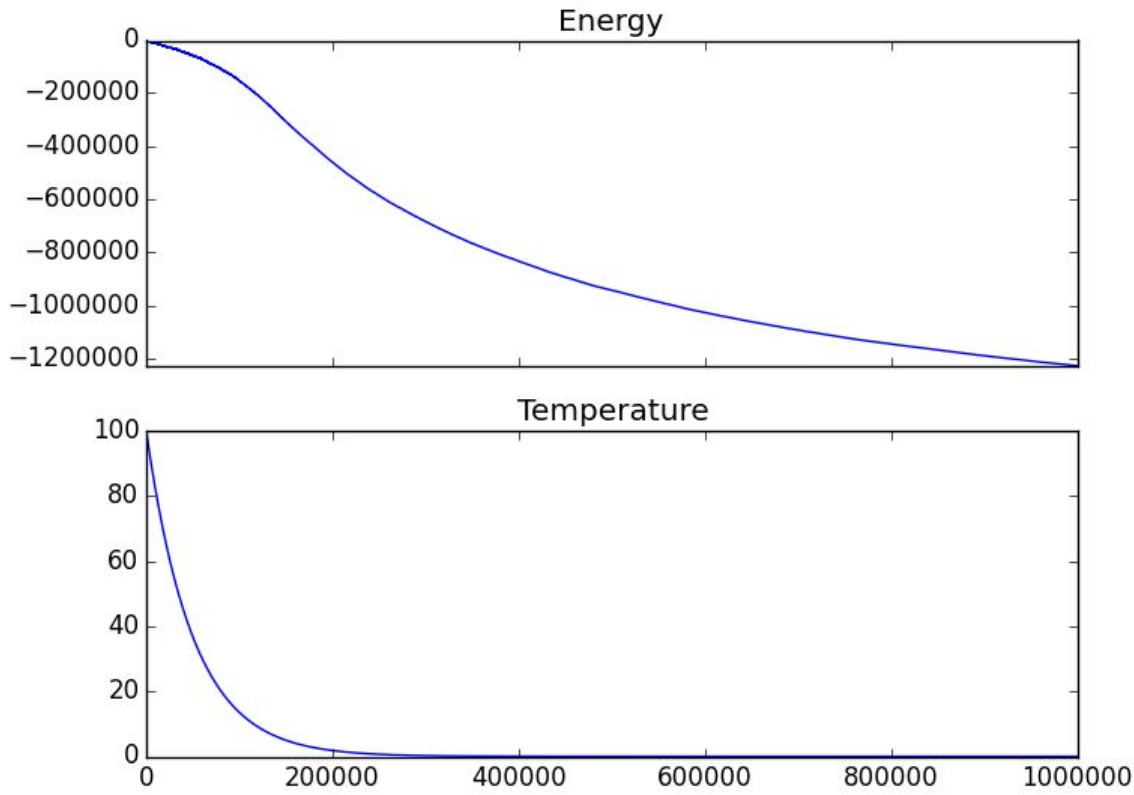
Przed wyżarzaniem:



Po wyżarzaniu:

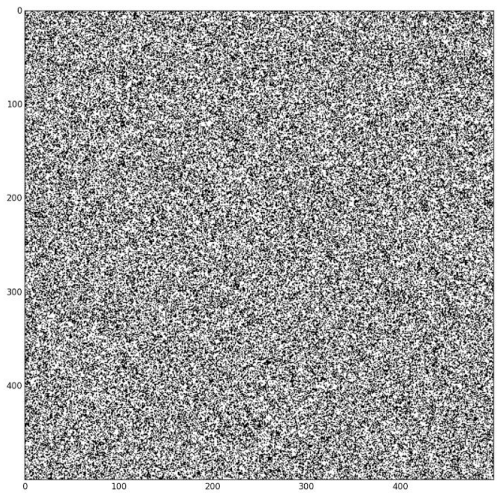


Energia i Temperatura:

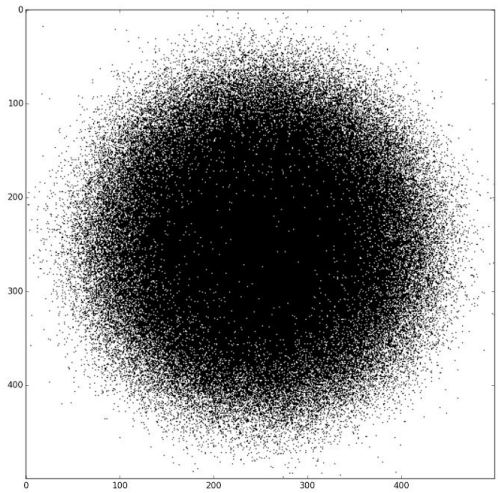


500x500, density=0.4, funkcja energii=energy2, funkcja sąsiedztwa=neighbour1, k=9000000

Przed wyżarzaniem:



Po wyżarzaniu:



Stany sąsiednie można generować obliczając energie wszystkich punktów, lub raz obliczyć energie wszystkich punków a następnie obliczać tylko różnice energii dla stanu sąsiedniego tj. różnice energii punktów zamienionych i ich sąsiadów.

Kod:

```
import math
import random
import copy
import matplotlib.pyplot as plt


def temp(T, n):
    return T * n


def distance(x1, y1, x2, y2):
    return math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)


def neighbour1(x, y, arr, n):
    for x1 in range(x - 1, x + 2):
        if x1 != x:
            yield x1 % n, y, arr[x1 % n][y]
    for y1 in range(y - 1, y + 2):
        if y1 != y:
            yield x, y1 % n, arr[x][y % n]


def neighbour2(x, y, arr, n):
```

```
    for x1 in range(x - 1, x + 2):
        for y1 in range(y - 1, y + 2):
            if x1 != x or y1 != y:
                yield x1 % n, y1 % n, arr[x1 % n][y1 % n]


def neighbour3(x, y, arr, n):
    for x1 in range(x - 6, x + 7):
        for y1 in range(y - 6, y + 7):
            if x1 != x or y1 != y:
                yield x1 % n, y1 % n, arr[x1 % n][y1 % n]


def neighbour4(x, y, arr, n):
    for x1 in range(x - 8, x + 9):
        for y1 in range(y - 8, y + 9):
            if x1 != x or y1 != y:
                yield x1 % n, y1 % n, arr[x1 % n][y1 % n]


def energy1(neighbour, d, x, y, n):
    en = 0.0
    for x1, y1, p in neighbour:
        if d == 0:
            if y1 < n / 2:
                en -= 1
            else:
                en += 1
        else:
            if y1 < n / 2:
                en += 1
            else:
                en -= 1
    return en


def energy2(neighbour, d, x, y, n):
    en = 0.0
    mid = n / 2
    if d == 0:
            en += distance(mid, mid, x, y)
    return en


def energy3(neighbour, d, x, y, n):
    en = 0.0
```

```
    if d == 0:
        for x1, y1, p in neighbour:
            if p == 0:
                en -= 1
    return en


def energy4(neighbour, d, x, y, n):
    en = 0.0
    for x1, y1, p in neighbour:
        if d == p:
            en += 1
        else:
            en -= 1
    return en


def energy5(neighbour, d, x, y, n):
    en = 0.0
    for x1, y1, p in neighbour:
        if d == p:
            en -= 1
        else:
            en += 1
    return en


def energy6(neighbour, d, x, y, n):
    en = 0.0
    for x1, y1, p in neighbour:
        if d == p:
            if x <= x1:
                en -= 1
            else:
                en += 1
        else:
            if x <= x1:
                en += 1
            else:
                en -= 1
    return en


def energy_dif(x1, y1, x2, y2, arr, new, neig, energ, n):
    en = 0.0
    en += energ(neig(x1, y1, arr, n), arr[x1][y1], x1, y1, n)
```

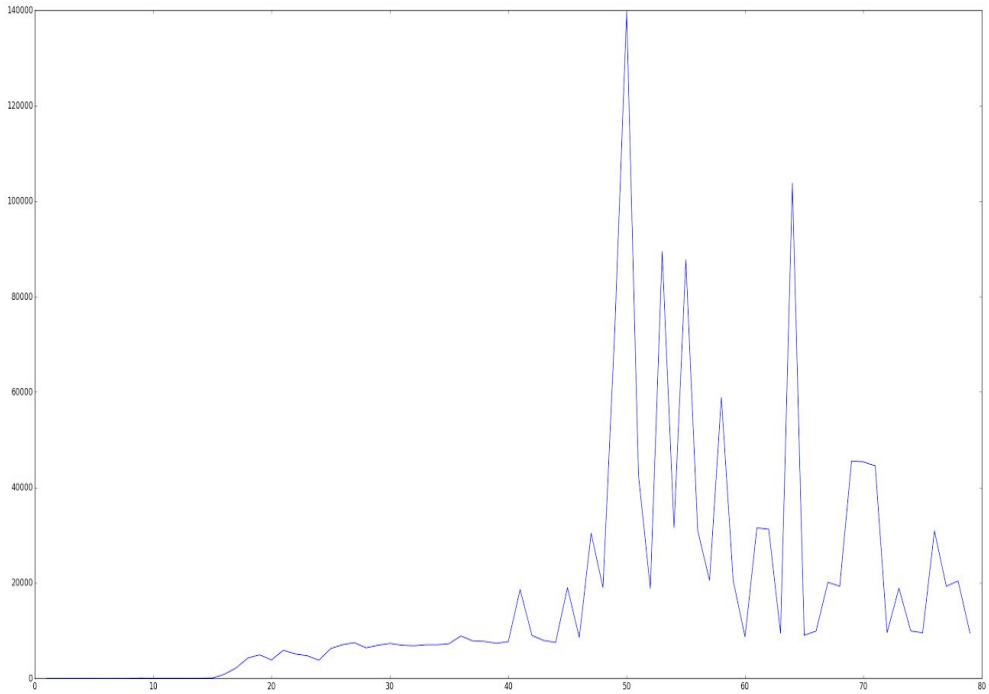
```
    en += energ(neig(x2, y2, arr, n), arr[x2][y2], x2, y2, n)
    en -= energ(neig(x1, y1, new, n), new[x1][y1], x1, y1, n)
    en -= energ(neig(x2, y2, new, n), new[x2][y2], x2, y2, n)
    return 2 * en


def calculate(arr, n, neig, energys):
    e = 0.0
    for i in range(n):
        for j in range(n):
            e += energys(neig(i, j, arr, n), arr[i][j], i, j, n)
    return e

def simple_swap(arr, n):
    new = [d[:] for d in arr]
    x1 = random.randrange(n)
    y1 = random.randrange(n)
    x2 = random.randrange(n)
    y2 = random.randrange(n)

    while (x1 == x2 and y1 == y2) or new[x1][y1] == new[x2][y2]:
        x2 = random.randrange(n)
        y2 = random.randrange(n)

    new[x1][y1], new[x2][y2] = new[x2][y2], new[x1][y1]
    return new, x1, y1, x2, y2

def sim_annealing(arr, n, neig=neighbour4, energys=energy1,
T=100.0, k=100000):
    energy_val = []
    temperatur = []
    current_solution = copy.deepcopy(arr)
    current_val = calculate(current_solution, n, neig, energys)
    best_solution = current_solution
    best_val = current_val
    for i in range(k):
        neighbour, x1, y1, x2, y2 = simple_swap(current_solution,
n)
        second_val = current_val - energy_dif(x1, y1, x2, y2,
current_solution, neighbour, neig, energys, n)
        if second_val < current_val:
            current_solution = neighbour
            current_val = second_val
        else:
            if math.exp((current_val - second_val) / T) >
random.random():
```

```
                current_solution = neighbour
                current_val = second_val
        if best_val > current_val:
            best_solution = current_solution
            best_val = current_val
        temperatur.append(T)
        T = temp(T, 0.99998)
        energy_val.append(current_val)
        if i % 1000 == 0:
            print("Current val:" + str(current_val) + " Best so far:
" + str(best_val) + " T: " + str(T))
    print("Calculated val: " + str(best_val))
    return best_solution, energy_val, temperatur

def main():
    T = 100.0
    k = 600000
    n = 500
    density = 0.5
    density = int(density * n ** 2)
    arr = [[1 for i in range(n)] for x in range(n)]
    for point in range(density):
        x = random.randrange(n)
        y = random.randrange(n)
        while arr[x][y] == 0:
            x = random.randrange(n)
            y = random.randrange(n)
        arr[x][y] = 0

    plt.figure(1)
    plt.imshow(arr, cmap=plt.get_cmap('gray'))
    solution, energy, temperature = sim_annealing(arr, n,
neighbour1, energy1, T, k)
    plt.figure(2)
    plt.imshow(solution, cmap=plt.get_cmap('gray'))

    f, (ax1, ax2,) = plt.subplots(2, sharex=True)
    ax1.axis([0, len(energy), min(energy), max(energy)])
    ax1.plot(range(len(energy)), energy)
    ax1.set_title('Energy')
    ax2.axis([0, len(temperature), 0, int(T)])
    ax2.plot(range(len(temperature)), temperature)
    ax2.set_title('Temperature')

    plt.show()
if (__name__ == "__main__"):
```

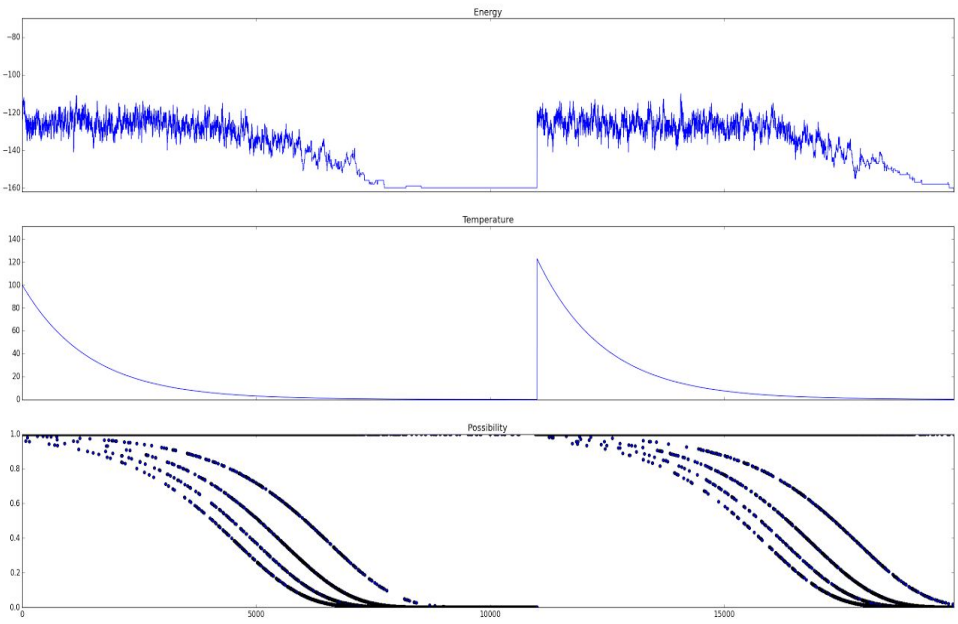
```
   main()
```

Zad 3

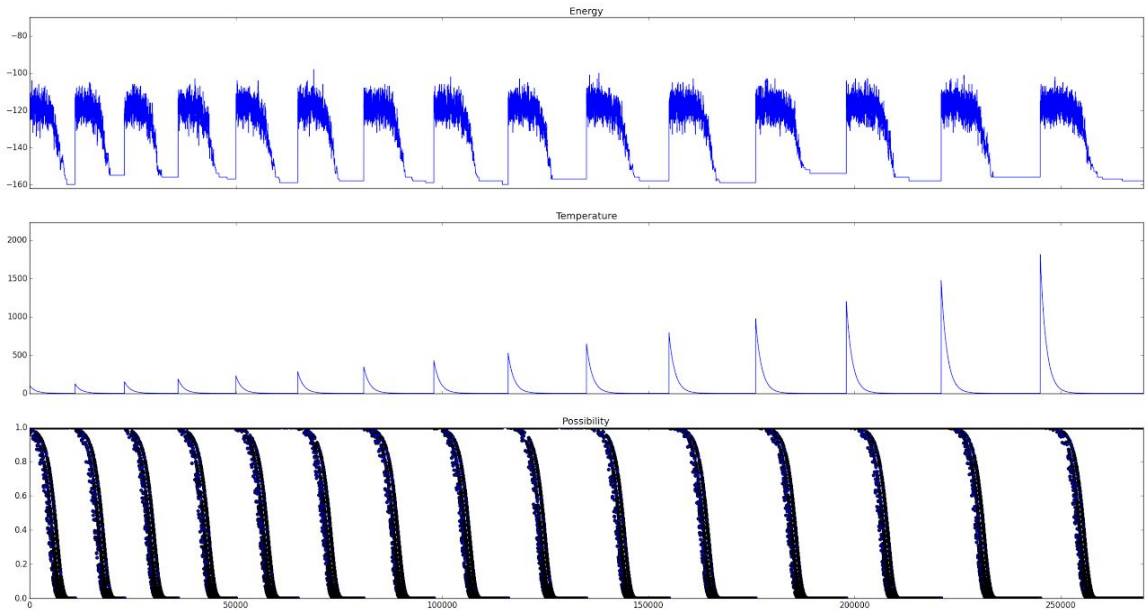
Ilość prób rozwiązania zależnie od ilości pustych pól:



Przykładowy wykresy dla rozwiązanego sudoku:



Oraz dla nieudanego rozwiązania:



Program nie rozwiązuje każdego sudoku.

Kod:

```
import math
import random
import os
import os.path
import matplotlib.pyplot as plt


def temp(T, n):
    return T * n


def calculate(arr):
    energy = 0
    for i in range(9):
        tmp = set()
```

```
        energy -= len(set(arr[i]))
        for j in range(9):
            tmp.add(arr[j][i])
        energy -= len((tmp))
    return energy


def simple_swap(arr, orig):
    new = [d[:] for d in arr]
    xb = random.randint(0, 2)
    yb = random.randint(0, 2)

    while ispossible(xb, yb, orig):
        xb = random.randint(0, 2)
        yb = random.randint(0, 2)

    ax = random.randint(xb * 3, xb * 3 + 2)
    ay = random.randint(yb * 3, yb * 3 + 2)

    while orig[ax][ay]:
        ax = random.randint(xb * 3, xb * 3 + 2)
        ay = random.randint(yb * 3, yb * 3 + 2)

    bx = random.randint(xb * 3, xb * 3 + 2)
    by = random.randint(yb * 3, yb * 3 + 2)

    while orig[bx][by] or (ax == bx and ay == by):
        bx = random.randint(xb * 3, xb * 3 + 2)
        by = random.randint(yb * 3, yb * 3 + 2)

    new[ax][ay], new[bx][by] = new[bx][by], new[ax][ay]
    return new


def ispossible(xb, yb, orig):
    count = 9
    for x in range(xb * 3, xb * 3 + 3):
        for y in range(yb * 3, yb * 3 + 3):
            if orig[x][y]:
                count -= 1
    return count <= 1


def sim_annealing(arr, orig, T, k):
    energy_val = []
    Temp =[]
```

```
    Poss = []
    current_solution = random_fill(arr)
    current_val = calculate(current_solution)
    best_solution = current_solution
    best_val = current_val
    for i in range(k):
        if best_val == -162:
            break
        neighbour = simple_swap(current_solution, orig)
        second_val = calculate(neighbour)
        if second_val < current_val:
            current_solution = neighbour
            current_val = second_val
        else:
            #print(math.exp((current_val - second_val) / T))
            if math.exp((current_val - second_val) / T) >
random.random():
                current_solution = neighbour
                current_val = second_val
        if best_val > current_val:
            best_solution = current_solution
            best_val = current_val
        Temp.append(T)
        T = temp(T, 0.9993)
        energy_val.append(current_val)
        Poss.append(math.exp((current_val - second_val) / T))

    return best_solution, energy_val, best_val, Temp, Poss


def read_from_file(f):
    arr = []
    orig = []
    for line in f:
        c = line.replace("x", "0")
        if len(c.split()) !=1:
            c = [int(a) for a in c.split()]
        else:
            c = [int(a) for a in line if a != "\n"]
        d = [(0 != q) for q in c]
        arr.append(c)
        orig.append(d)
    return arr, orig

def generate(f, i):
    arr = []
```

```
    orig = []
    for line in f:
        c = line.replace("x", "0")
        if len(c.split()) !=1:
            c = [int(a) for a in c.split()]
        else:
            c = [int(a) for a in line if a != "\n"]
        d = [(0 != q) for q in c]
        arr.append(c)
        orig.append(d)
    for x in range(i):
        x = random.randrange(9)
        y = random.randrange(9)
        while arr[x][y] == 0:
            x = random.randrange(9)
            y = random.randrange(9)
        arr[x][y] = 0
        orig[x][y] = False
    return arr, orig

def write_to_file(f, solution):
    for i in range(9):
        for j in range(9):
            f.write(str(solution[i][j]))
        f.write('\n')


def random_fill(arr):
    new = [d[:] for d in arr]
    i = [k for k in range(1, 10)]
    random.shuffle(i)
    for xb in range(3):
        for yb in range(3):
            for x in range(xb * 3, xb * 3 + 3):
                for y in range(yb * 3, yb * 3 + 3):
                    if new[x][y] == 0:
                        for j in i:
                            if is_not_in_block(j, new, xb, yb):
                                new[x][y] = j
                                break
            random.shuffle(i)
    return new


def is_not_in_block(i, arr, xb, yb):
    for x in range(xb * 3, xb * 3 + 3):
```

```
        for y in range(yb * 3, yb * 3 + 3):
            if arr[x][y] == i:
                return False

    return True


def main():
    T = 100.0
    k = 10000
    i = 1
    m = 0

    kx =[]
    for name in os.listdir('./gen'):
        if os.path.isfile('./gen/' + name):
            for x in range(1,80):
                total = 0
                best_value = 0
                energy_val = []
                tempe = []
                poss =[]
                with open('./gen/' + name, 'r') as f:
                    arr, orig = generate(f, x)
                while best_value != -162 and m < 10:
                    m += 1
                    solution, energy_val1, best_value, tempe1,
poss1 = sim_annealing(arr, orig, T, k)
                    energy_val.extend(energy_val1)
                    total += len(energy_val1)
                    k += 1000
                    T *= 1.23
                kx.append(total)
                print("Calculated val: " + str(best_value) + ",
Solved should be -162. Solved in " + str(total)+" steps and
"+str(m-1)+ " resets")
                k = 10000
                T = 100.0
                m = 0
    plt.plot(range(1,80), kx)
    plt.show()

    for name in os.listdir('./test'):
        if os.path.isfile('./test/' + name):
            best_value = 0
            energy_val = []
```

```
            tempe = []
            poss =[]
            print("Next one:\n")
            with open('./test/' + name, 'r') as f:
                arr, orig = read_from_file(f)
            for p in range(9):
                print(arr[p])
            while best_value != -162 and m < 15:
                m += 1
                k += 1000
                solution, energy_val1, best_value, tempe1, poss1 =
sim_annealing(arr, orig, T, k)
                energy_val.extend(energy_val1)
                tempe.extend(tempe1)
                poss.extend(poss1)
                T *= 1.23
            print("Calculated val: " + str(best_value) + ", Solved
should be -162. Solved in " + str(len(energy_val))+" steps and
"+str(m-1)+ " resets")
            f, (ax1, ax2, ax3) = plt.subplots(3, sharex=True)
            ax1.axis([0,len(energy_val),-162,-70])
            ax1.plot(range(len(energy_val)), energy_val)
            ax1.set_title('Energy')
            ax2.axis([0, len(energy_val), 0, T])
            ax2.plot(range(len(tempe)), tempe)
            ax2.set_title('Temperature')
            ax3.axis([0,len(energy_val),0,1])
            ax3.scatter(range(len(poss)), poss)
            ax3.set_title('Possibility')
            k = 10000
            T = 100.0
            m = 0
            with open(str(i) + "_solution.txt", 'w') as f:
                write_to_file(f, solution)
            i += 1
            for q in range(9):
                print(solution[q])
    plt.show()

if(__name__ =="__main__"):
    main()
```