Paweł Prus

IDE: Microsoft Visual Studio Code
Procesor: AMD Ryzen 7 6800h
Język: Python 3.10.12
Biblioteki:
numpy jako np
matplotlib.pyplot jako plt
time

Sprawozdanie z zastosowania algorytmu symulowanego wyżarzania

1. Opis problemu

1.1. W ramach projektu rozważono trzy różne problemy optymalizacyjne, do których zastosowano algorytm symulowanego wyżarzania:

1.1.1. Problem komiwojażera (TSP) - plik *zad1_1.ipynb*: Celem jest znalezienie najkrótszej trasy łączącej wszystkie podane punkty, zaczynając i kończąc trasę w tym samym punkcie.

1.1.2. Segregacja punktów białych i czarnych - plik *zad1_2.ipynb*: Zadanie polega na uporządkowaniu punktów (reprezentujących piksele obrazu) w taki sposób, aby punkty białe i czarne były rozdzielone według zadanej funkcji sąsiedztwa.

1.1.3. Rozwiązanie sudoku - plik *zad1_3.ipynb*: Celem jest znalezienie rozwiązania planszy sudoku, zachowując zasady gry.

2. Opis implementowanych funkcji

2.1. Plik zad1_1.ipynb

2.1.1. *distance(point_a, point_b)*: Oblicza odległość euklidesową między dwoma punktami.

2.1.2. *generate_circle_point(centerpoint, radius)*: Generuje losowy punkt na okręgu o zadanym środku i promieniu.

2.1.3. *generate_points(no_points, centerpoint, no_groups, radius=1)*: Generuje punkty wokół zadanego centrum w grupach, gdzie każda grupa ma oddzielny promień.

2.1.4. *simulated_annealing(points, iterations=3000, T=5000, cooling_rate=0.995)*: Implementuje algorytm symulowanego wyżarzania dla problemu komiwojażera.

2.2. Plik zad1_2.ipynb

2.2.1. *generate_random_binary_image(n, delta, indices=None)*: Generuje losowy obraz binarny o rozmiarze n x n, gdzie delta określa stosunek liczby pikseli czarnych do całkowitej liczby pikseli.

2.2.2. Funkcje sąsiedztwa

2.2.2.1. *neighbourhood_energy_vert* –funkcja oblicza sumę energii punktu jako sumę jego prawego i lewego sąsiada (funkcja dąży do tego, aby każdy czarny punkt miał na prawo i na lewo punkt biały)

2.2.2.2. *neighbourhood_energy_hor* – funkcja oblicza sumę energii punktu jako sumę jego górnego i dolnego sąsiada (funkcja dąży do tego, aby każdy czarny punkt miał na górze i na dole punkt biały)

2.2.2.3. *neighbourhood_energy_vert_stripes* – funkcja oblicza sumę energii punktu jako sumę jego prawego i lewego sąsiada oraz ich górnych i dolnych sąsiadów (funkcja dąży do tego, aby każdy czarny punkt miał na prawo i lewo punkt biały, oraz żeby ich górni i dolni sąsiedzi byli biali)

2.2.2.4. *neighbourhood_energy_hor_stripes* – funkcja oblicza sumę energii punktu jako sumę jego górnego i dolnego sąsiada oraz ich prawych i lewych sąsiadów

(funkcja dąży do tego, aby każdy czarny punkt miał na górze i na dole punkt biały, oraz żeby ich prawi i lewi sąsiedzi byli biali)

2.2.2.5. *neighbourhood_energy_divide* – funkcja oblicza sumę energii punktu jako sumę wszystkich punktów bezpośrednio z nim graniczących (funkcja dąży do tego, żeby w odległości 1 każdy punkt czarny miał tylko punkty białe)

2.2.2.6. *neighbourhood_energy_combine* – funkcja oblicza sumę energii punktu jako ujemną sumę wszystkich punktów bezpośrednio z nim graniczących (funkcja dąży do tego, żeby w odległości 1 każdy punkt czarny miał tylko punkty czarne)

2.2.3. *whole_energy(image, neighbourhood_energy_function, matrix=False)*: Oblicza całkowitą energię obrazu binarnego w zależności od zastosowanej funkcji sąsiedztwa.

2.2.4. *simulated_annealing_image(image, neighbourhood_energy_function, iterations=5000, T=3000, cooling_rate=0.995)*: Implementuje algorytm symulowanego wyżarzania do segregacji punktów białych i czarnych.

2.3. Plik zad1_3.ipynb

2.3.1. *draw_sudoku(matrix, edge_color1='black', edge_color2='black')*: Rysuje planszę sudoku na podstawie podanej macierzy.

2.3.2. *sudoku_matrix(filename=None)*: Tworzy macierz reprezentującą planszę sudoku na podstawie pliku lub zwraca pustą macierz, jeśli plik nie jest podany.

2.3.3. *fill_matrix(matrix)*: Wypełnia puste pola w macierzy sudoku losowymi wartościami, zachowując zasady gry.

2.3.4. *energy(matrix, point, tellrange=False)*: Oblicza energię danego punktu w planszy sudoku, z uwzględnieniem powtarzających się wartości w wierszach, kolumnach i kwadratach 3x3.

2.3.5. *whole_energy(matrix, matrix_gen=False)*: Oblicza całkowitą energię planszy sudoku na podstawie zadanej macierzy.

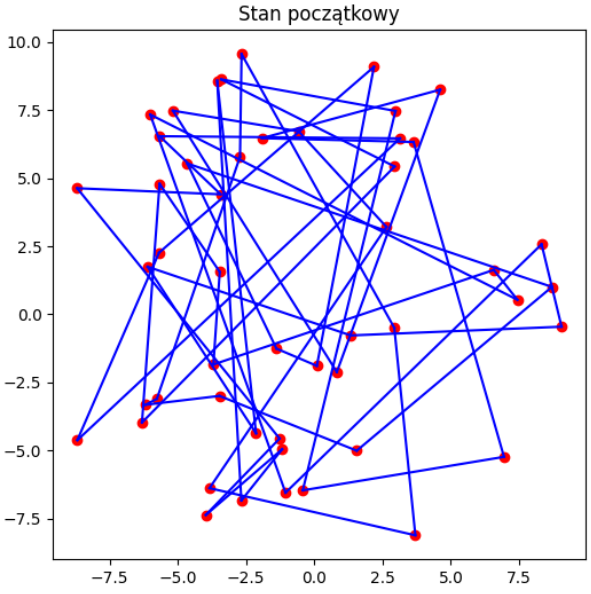
2.3.6. *simulated_annealing_sudoku(matrix, good_points, iterations=1_000, T=5_000, cooling_rate=0.995)*: Implementuje algorytm symulowanego wyżarzania w celu rozwiązania sudoku.

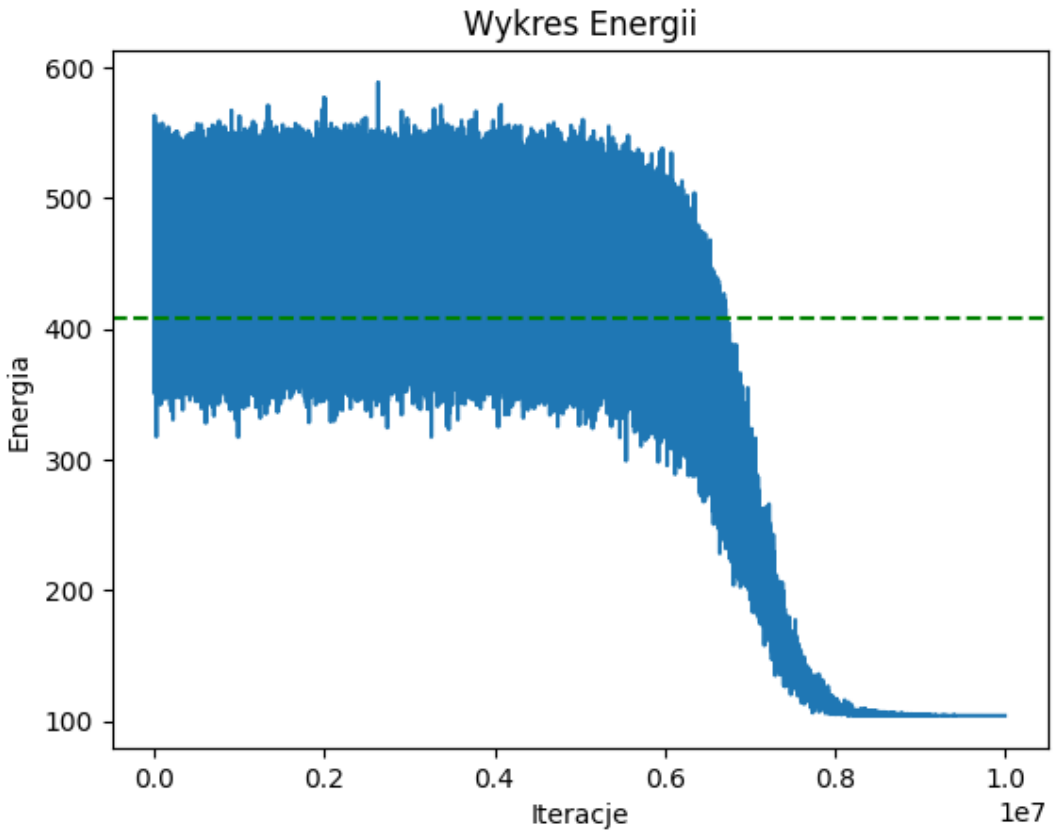
3. Analiza wyników

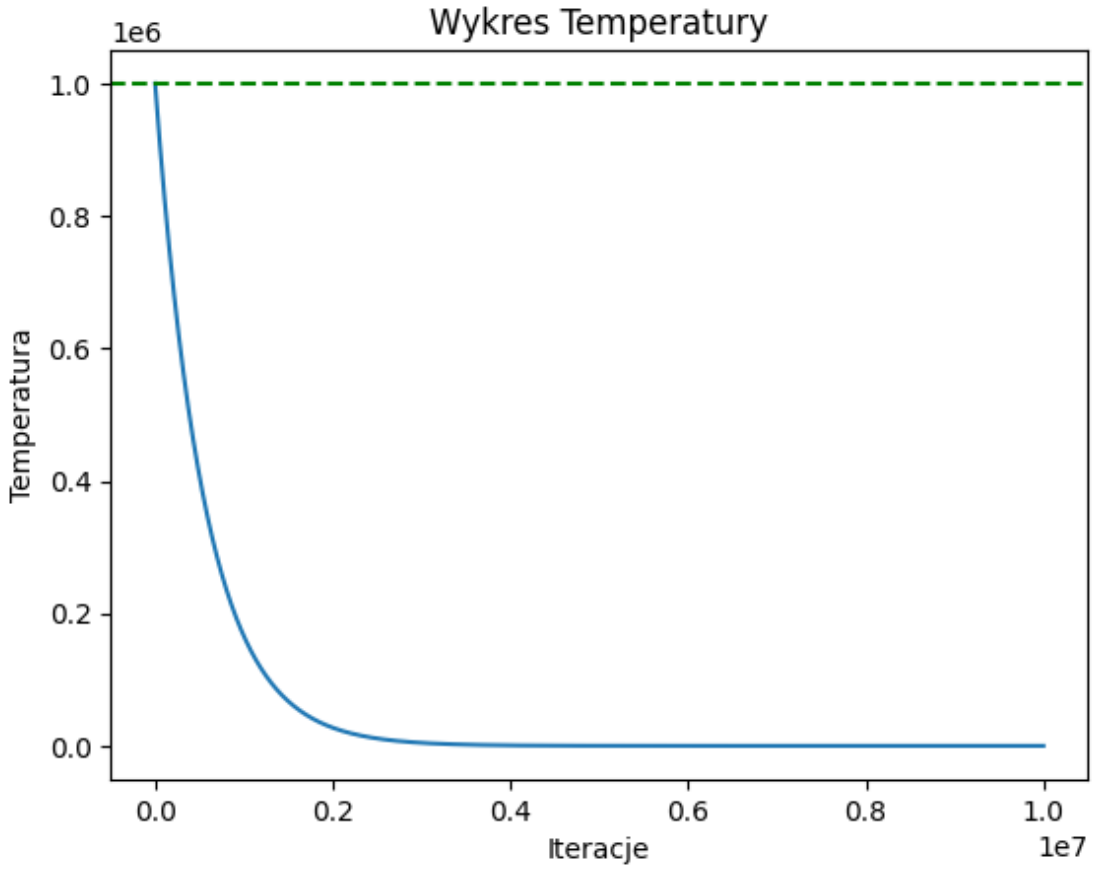
3.1. Plik *zad1_1.ipynb*

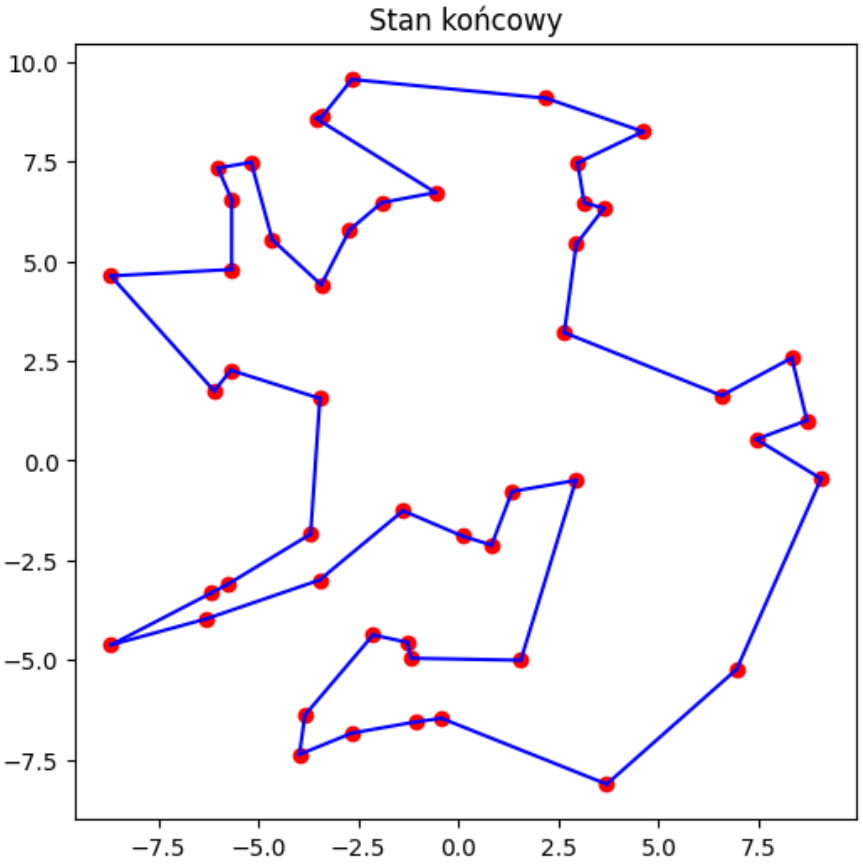
Parametr n oznacza liczbę punktów, a parametr g oznacza liczbę grup, parametr i oznacza liczbę iteracji, parametr t oznacza temperaturę początkową, a parametr c oznacza stopień chłodzenia

3.1.1. n = 50, g = 1, i = 10000000, t = 1000000, c = 0.9999982



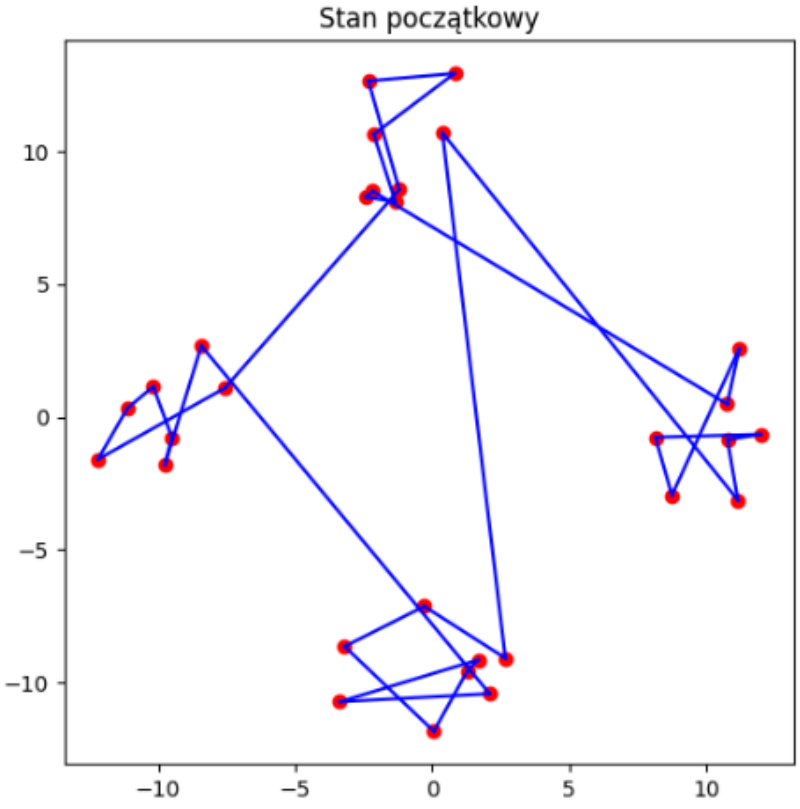

Wykres Energii
600
500
400
300
200
100
Energia
0.0
0.2
0.4
0.6
0.8
1.0
1e7
Iteracje

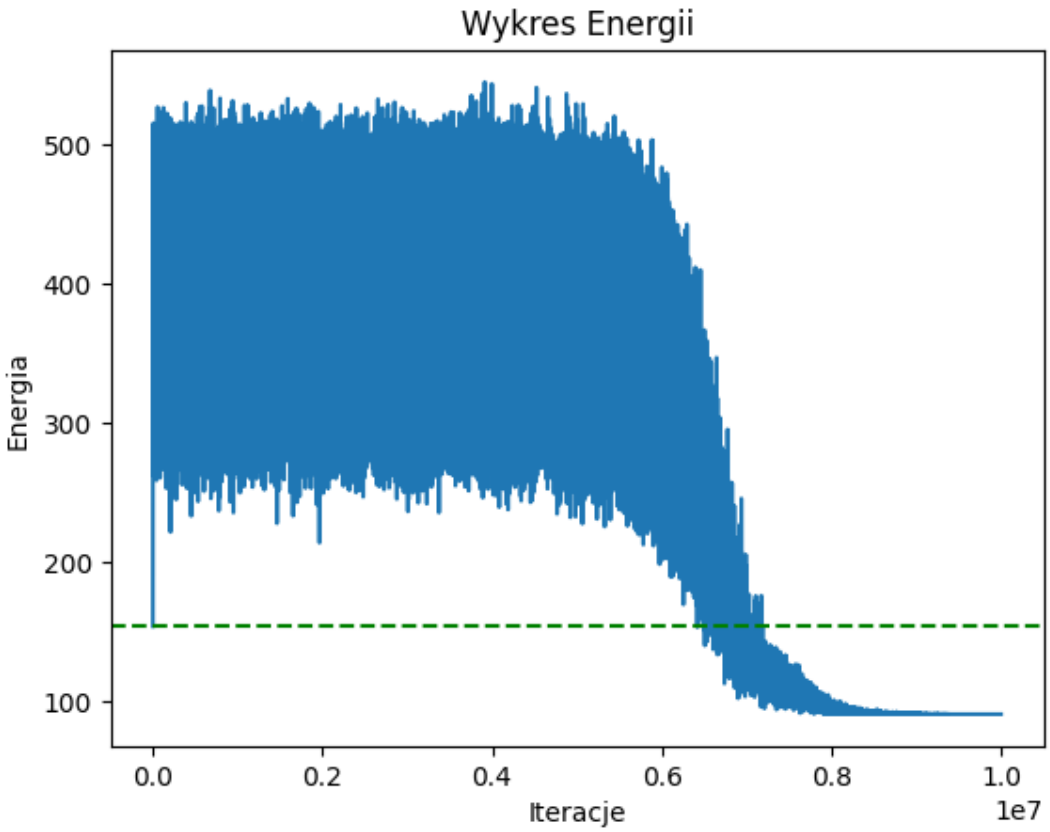

Wykres Temperatury
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Temperatura
0.0
0.2
0.4
0.6
0.8
1.0
1e7
Iteracje

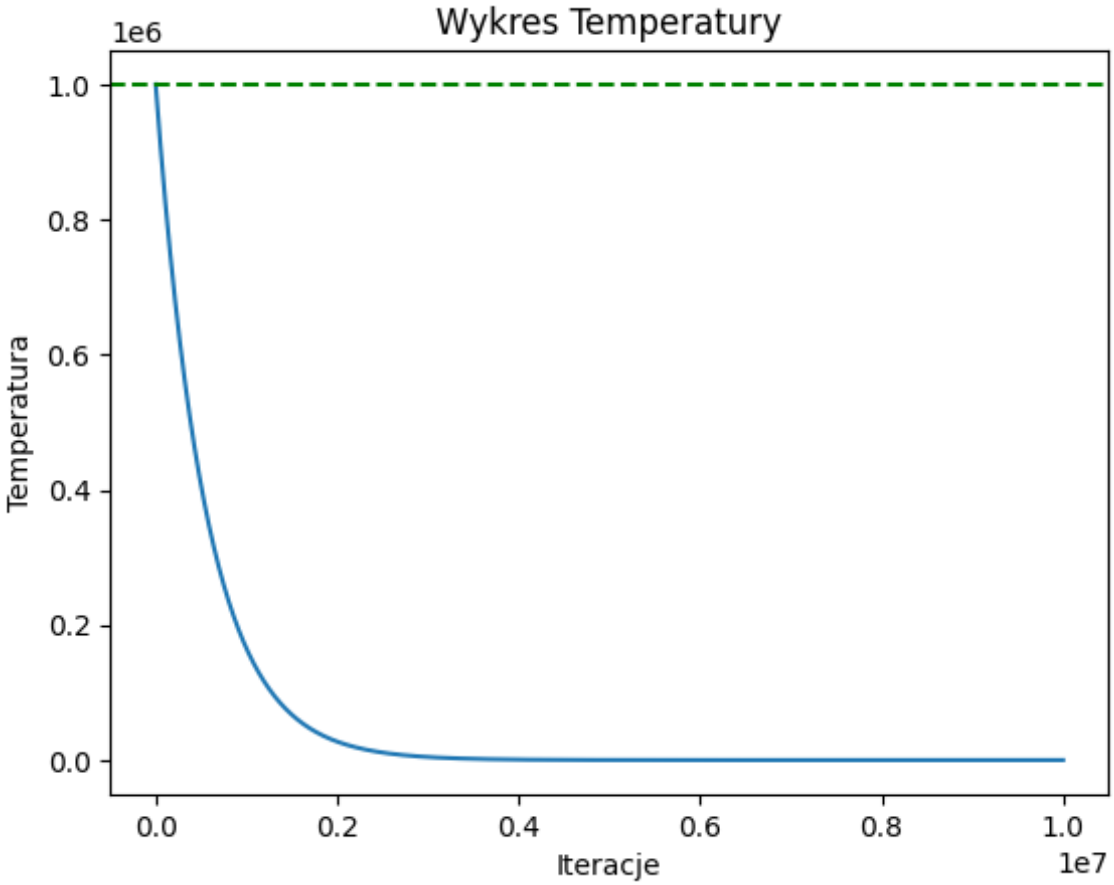


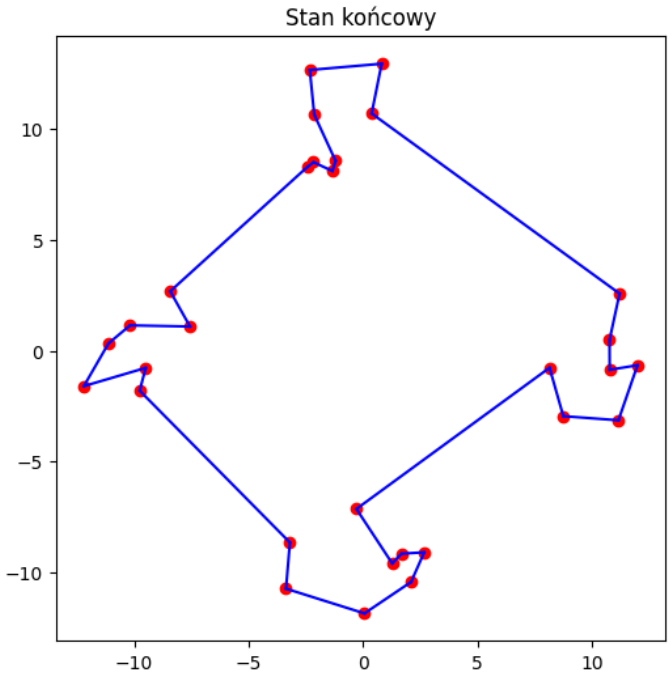
Ścieżka optymalna została odnaleziona w czasie 350 sekund

3.1.2.n = 30, g = 4, i = 10000000, t = 1000000, c = 0.9999982



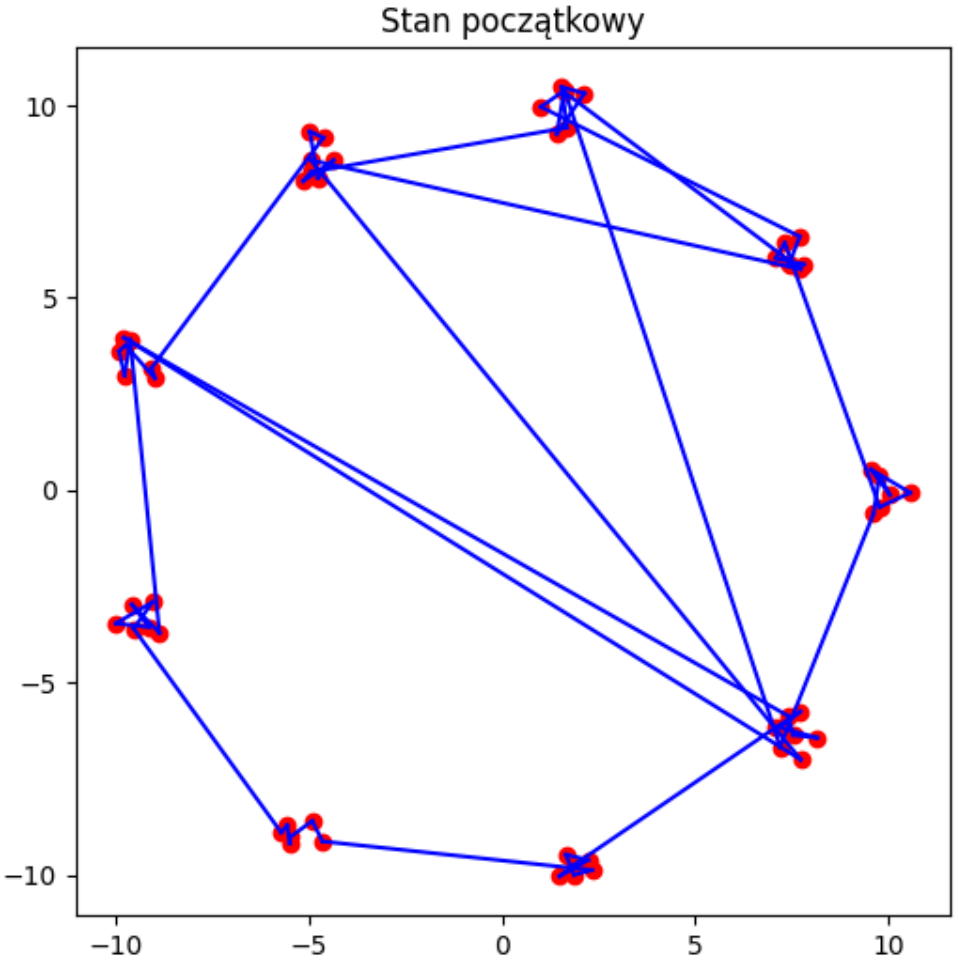

Wykres Energii
Energia
500
400
300
200
100
0.0
0.2
0.4
0.6
0.8
1.0
1e7
Iteracje

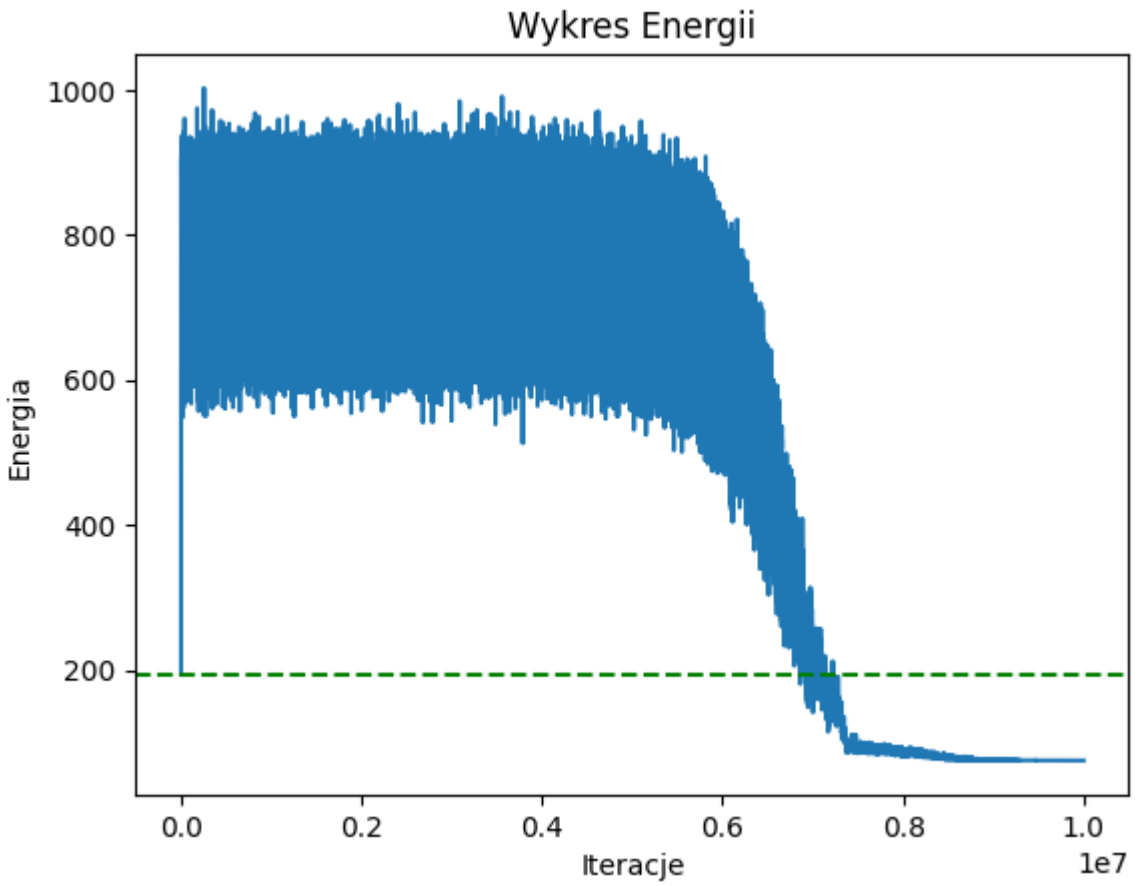

Wykres Temperatury
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Temperatura
0.0
0.2
0.4
0.6
0.8
1.0
1e7
Iteracje

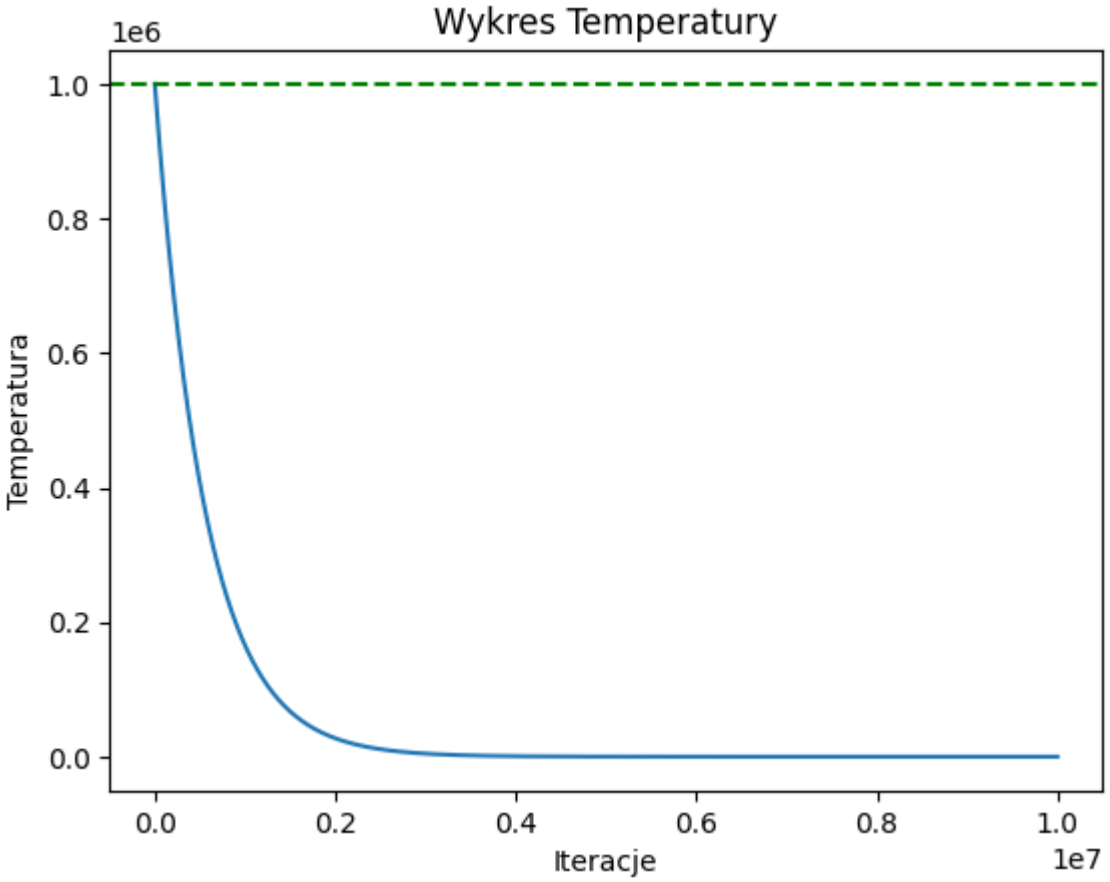


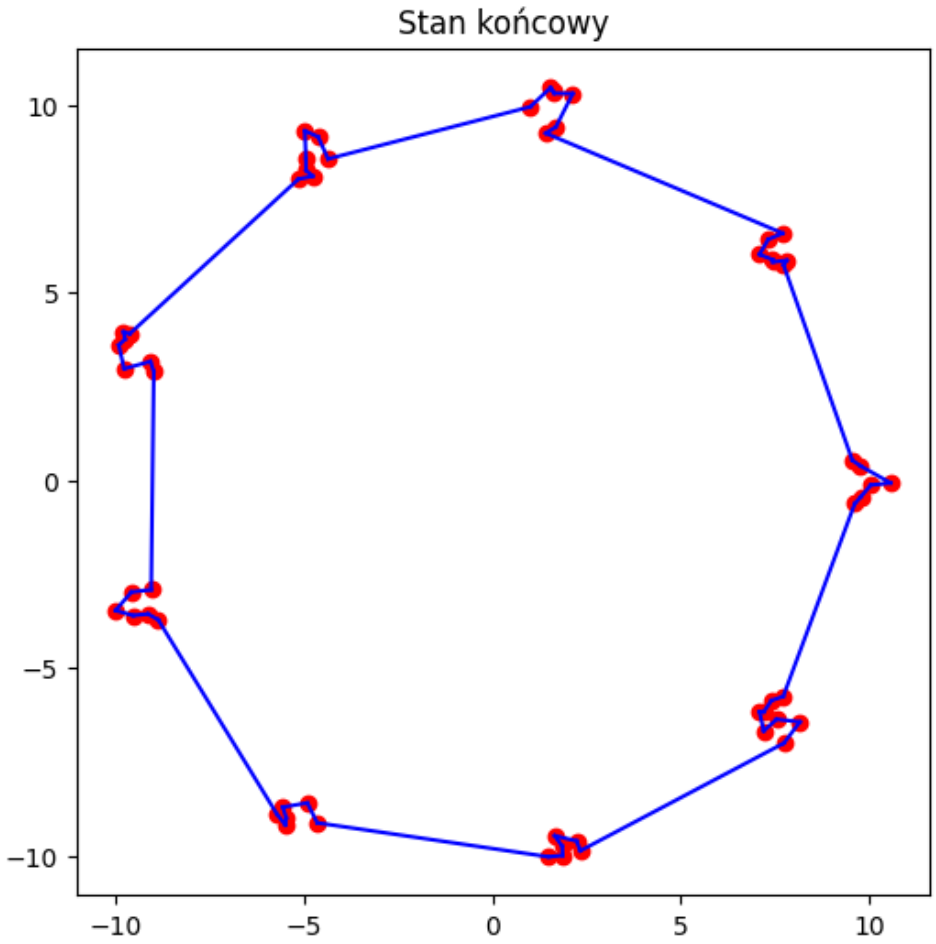
Ścieżka optymalna została odnaleziona w czasie 345 sek

3.1.3.n = 60, g = 9, i = 10000000, t = 1000000, c = 0.9999982




Wykres Energii
1000
800
600
400
200
Energia
0.0
0.2
0.4
0.6
0.8
1.0
1e7
Iteracje


Wykres Temperatury
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Temperatura
0.0
0.2
0.4
0.6
0.8
1.0
1e7
Iteracje

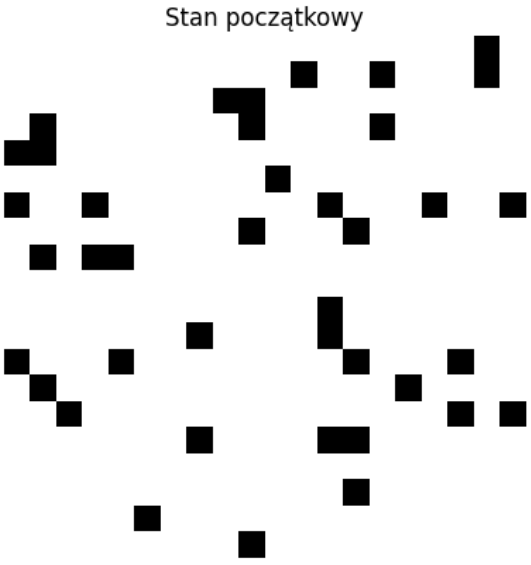


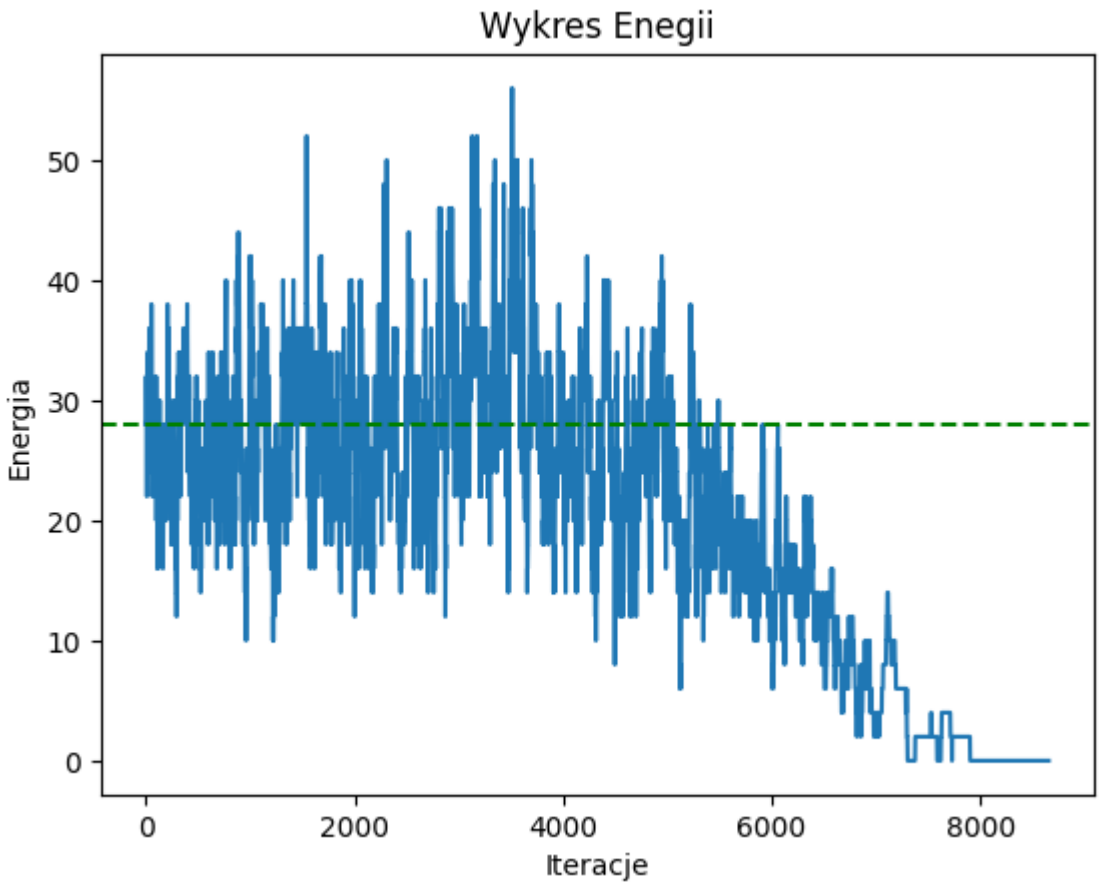
Ścieżka optymalna została odnaleziona w czasie 330 sek

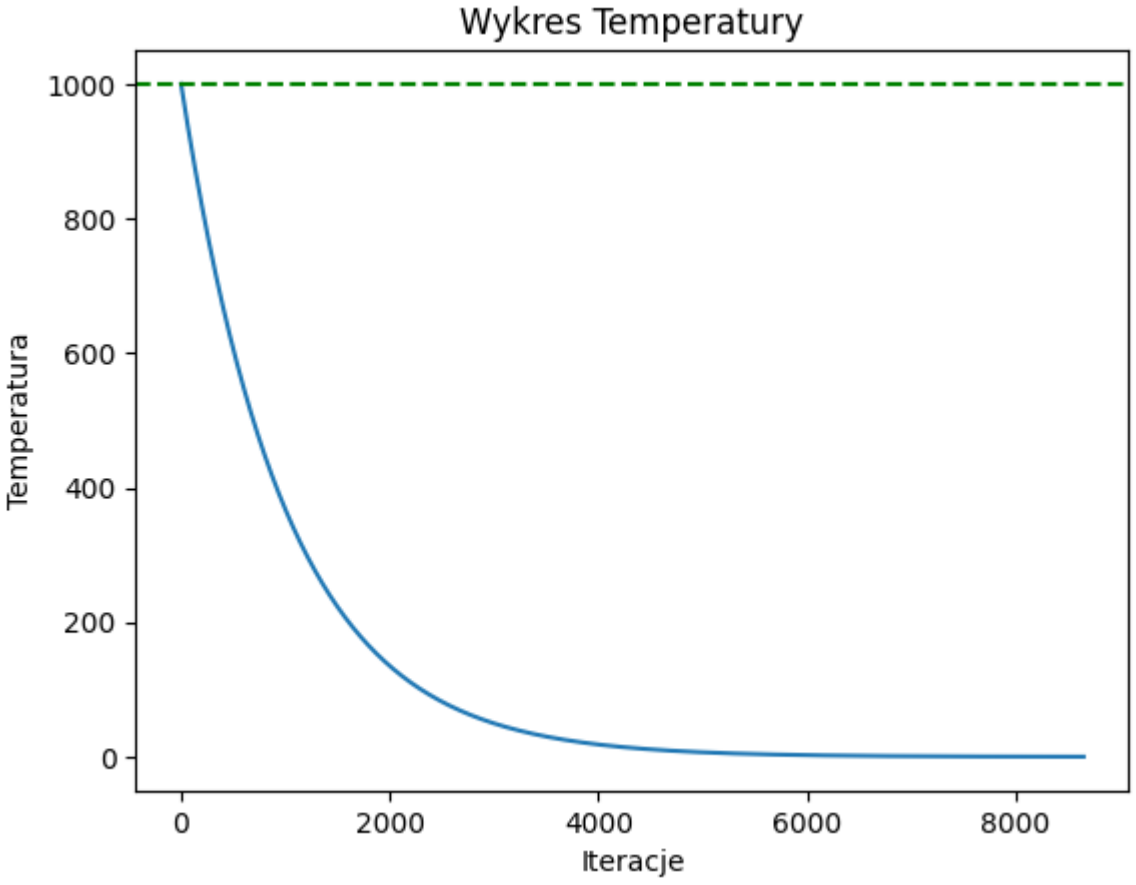
3.2. Plik *zad1_2.ipynb*

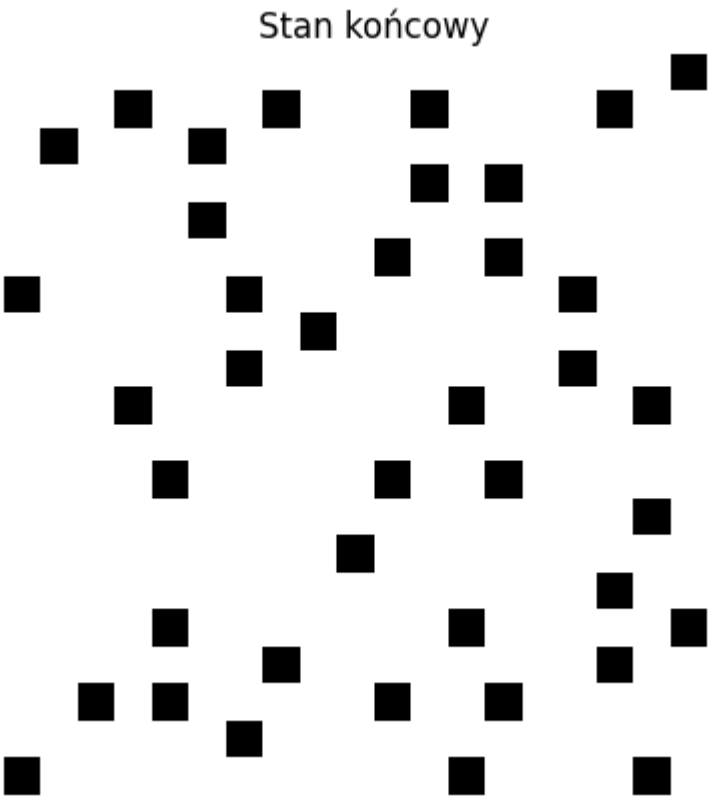
Parametr n oznacza szerokość i wysokość obrazu, delta oznacza zagęszczenie czarnych pikseli, a f oznacza funkcję generującą sąsiedztwo, parametr i oznacza liczbę iteracji, parametr t oznacza temperaturę początkową, a parametr c oznacza stopień chłodzenia

3.2.1.n = 20, δ = 0.1, f = *neighbourhood_energy_divide*, i = 100000, t = 1000, c = 0.999



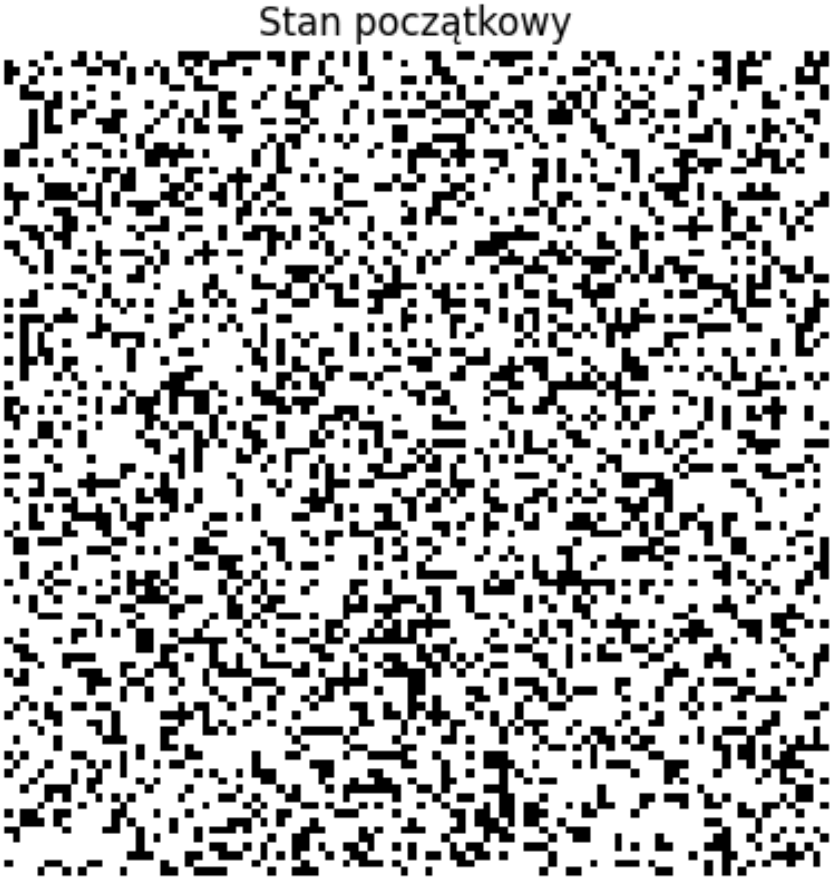

Wykres Enegii
Energia
Iteracje
0
10
20
30
40
50
0
2000
4000
6000
8000

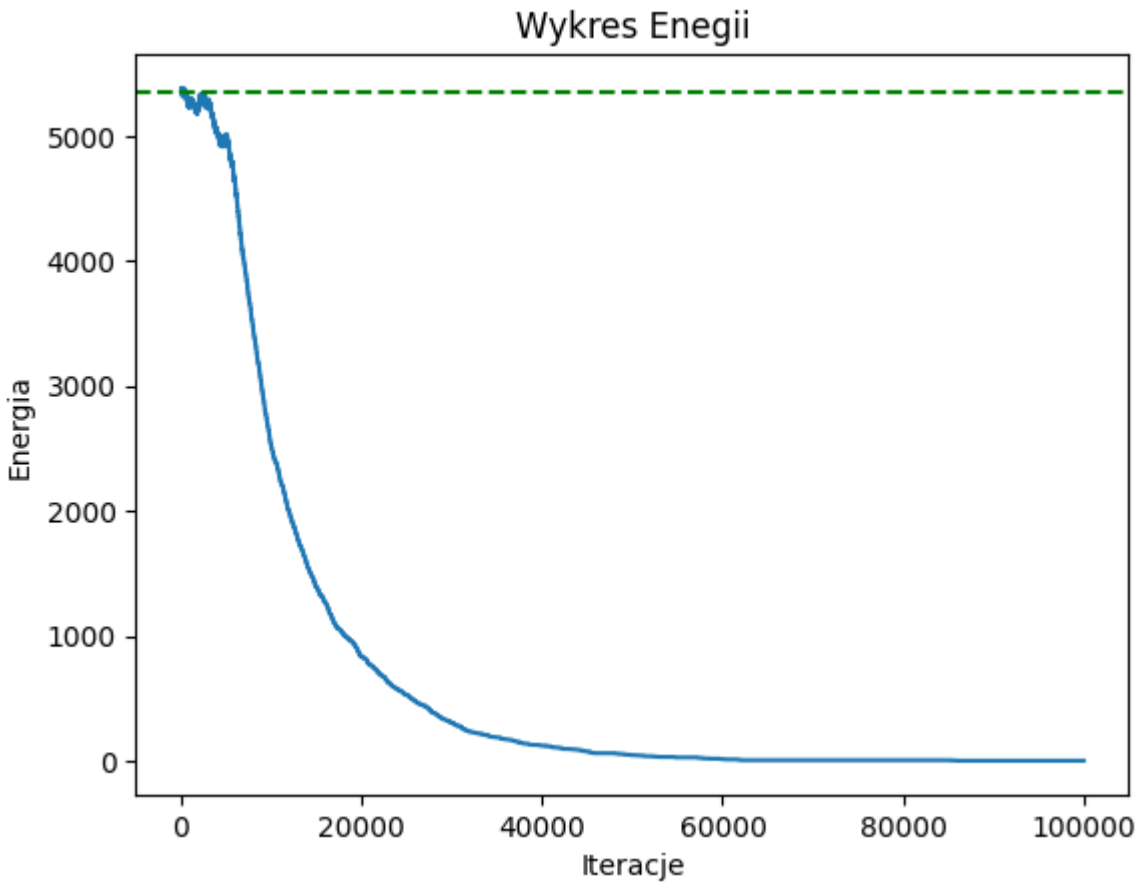

Wykres Temperatury
Temperatura
Iteracje
0
200
400
600
800
1000
0
2000
4000
6000
8000

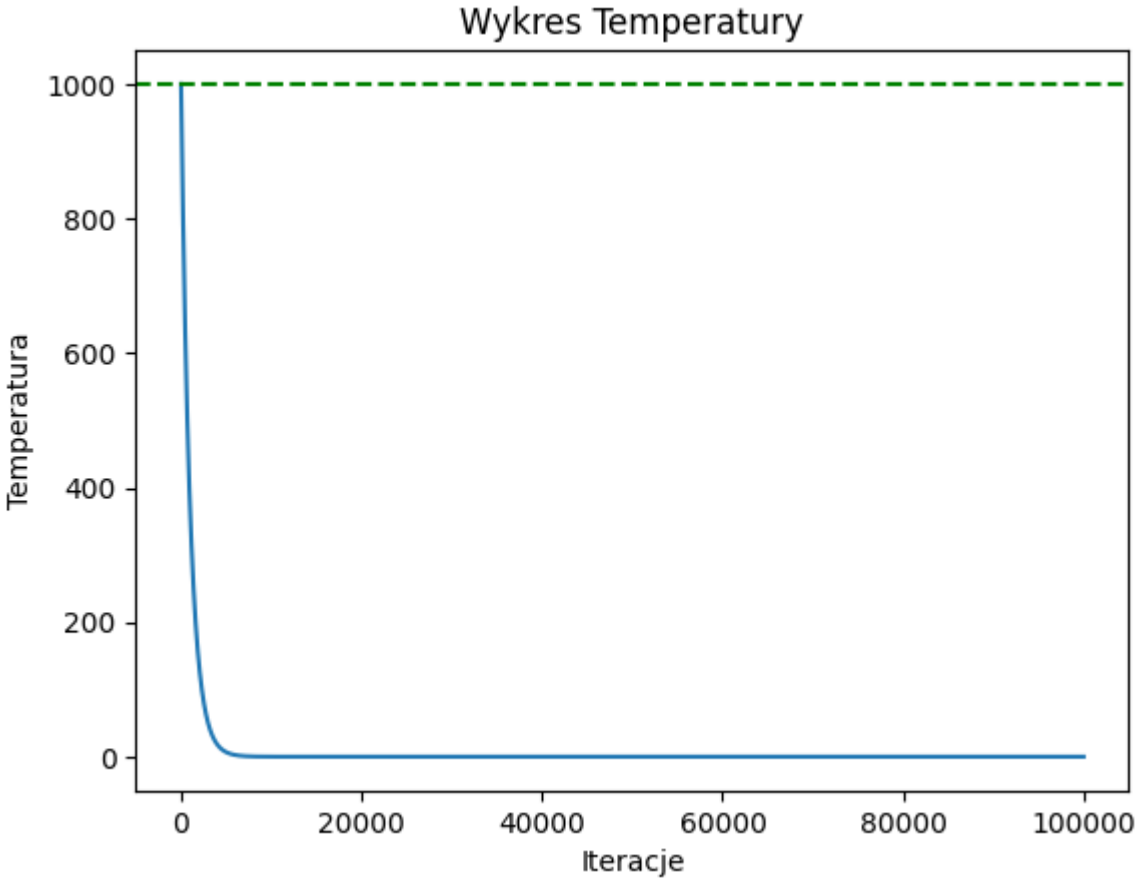


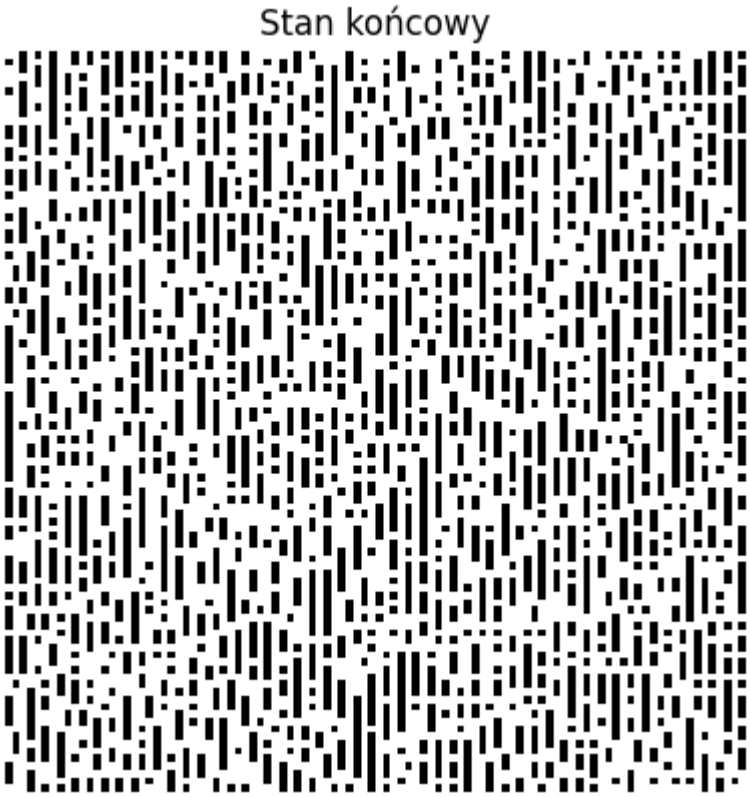
Stan optymalny został znaleziony w czasie 1.25 sek

3.2.2.n = 100, δ = 0.3, f = *neighbourhood_energy_vert_stripes*, i = 100000, t = 1000, c = 0.999



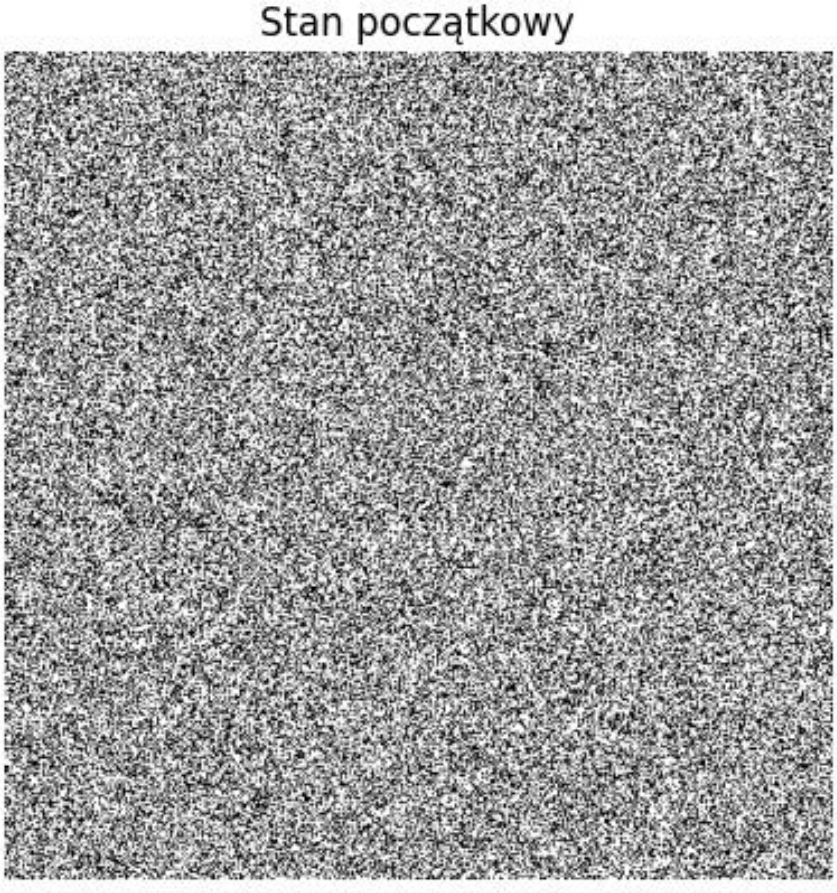

Wykres Enegii
Energia
5000
4000
3000
2000
1000
0
0
20000
40000
60000
80000
100000
Iteracje

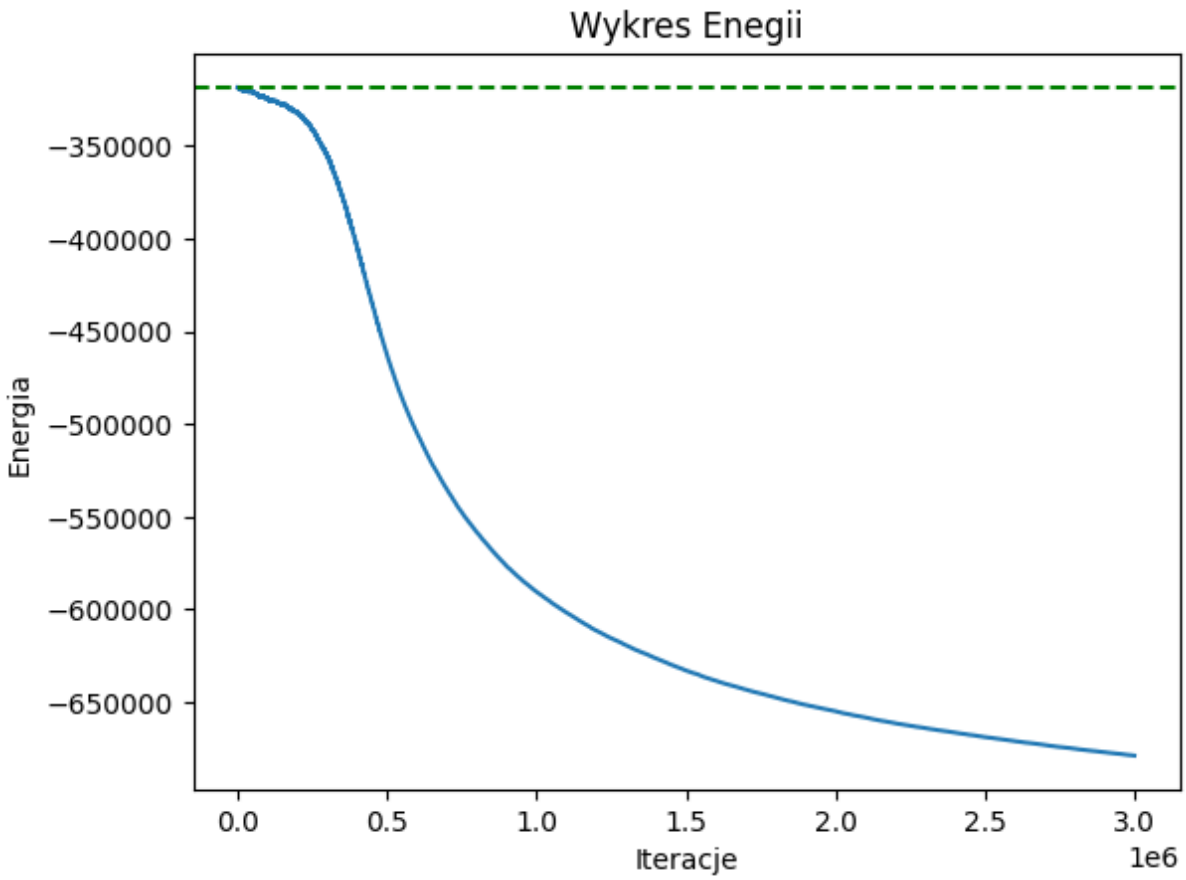

Wykres Temperatury
Temperatura
1000
800
600
400
200
0
0
20000
40000
60000
80000
100000
Iteracje

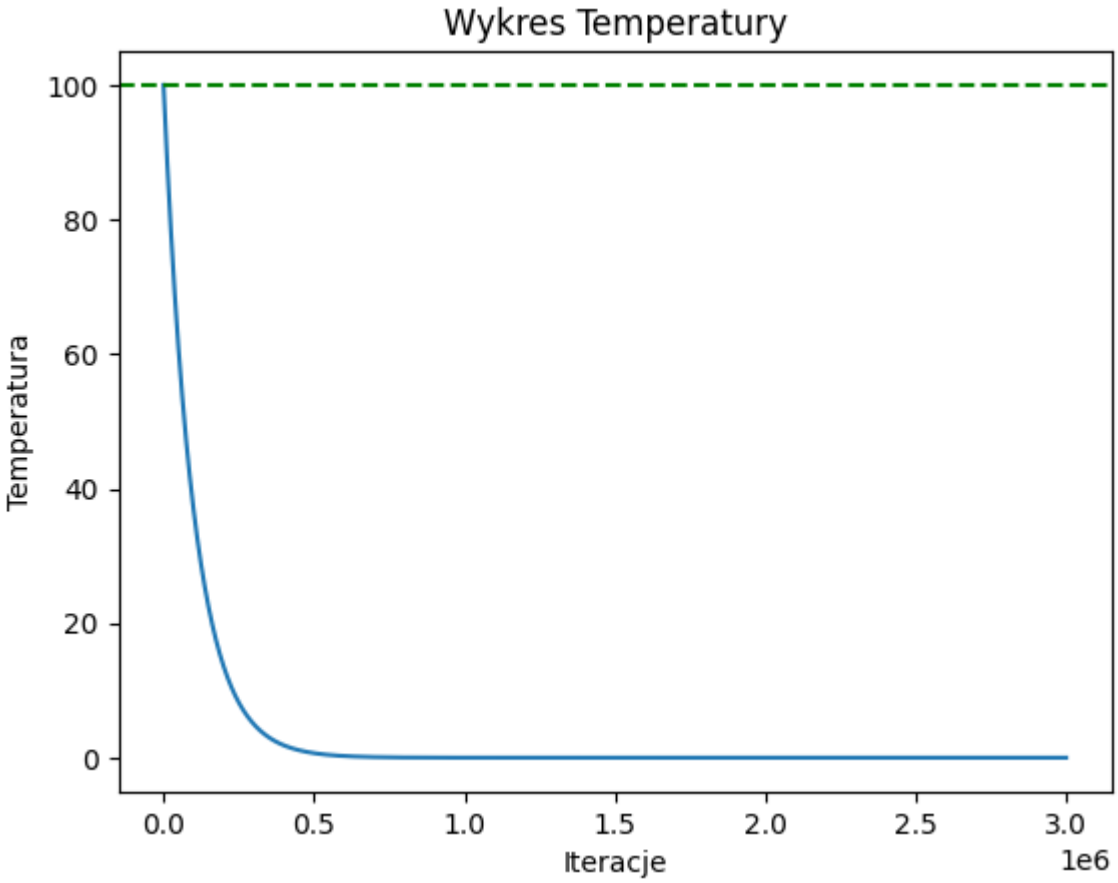


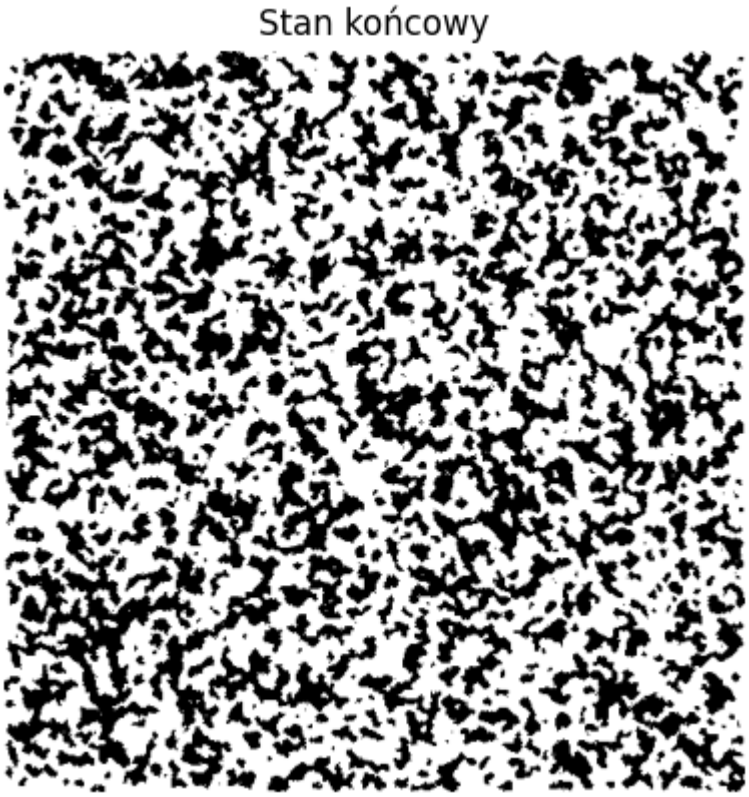
Stan optymalny został odnaleziony w 7.37 sek

3.2.3.n = 500, δ = 0.4, f = *neighbourhood_energy_combine*, i = 3000000, t = 100, c = 0.99999



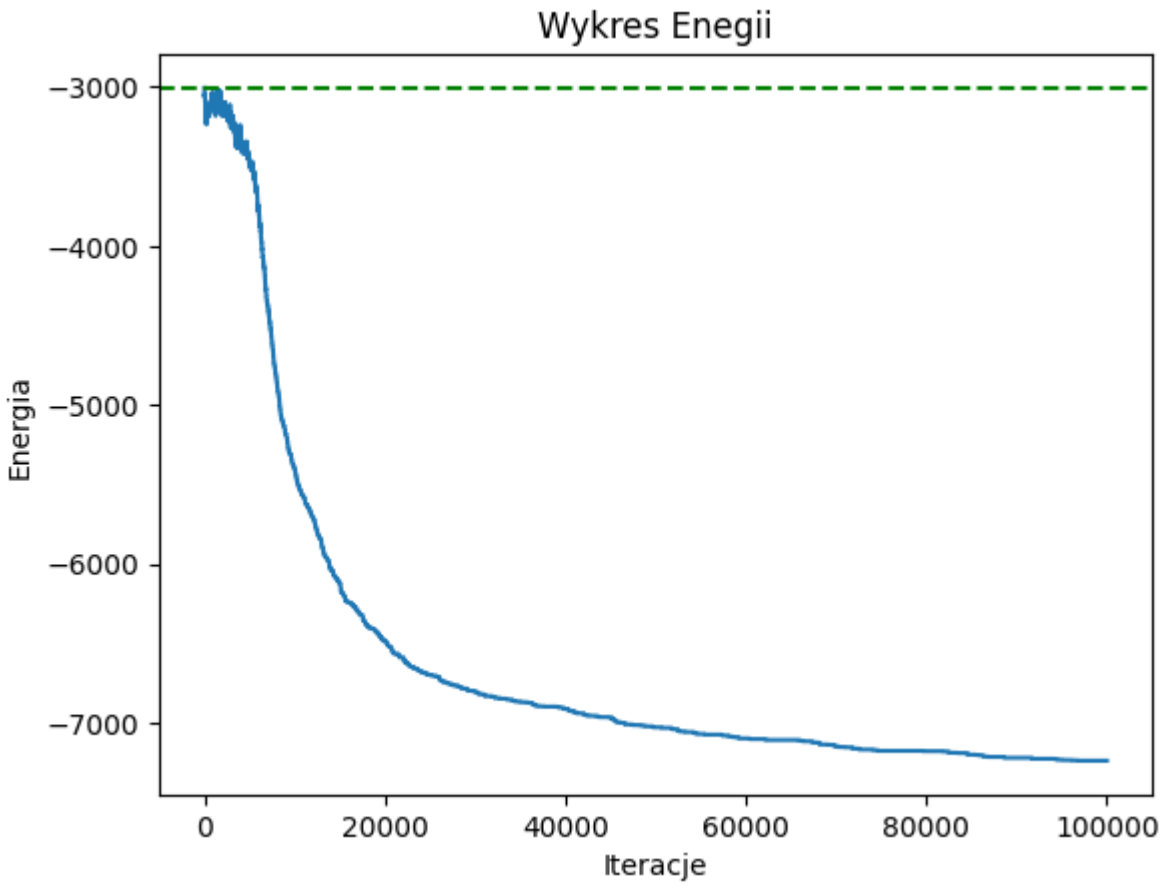

Wykres Enegii
−350000
−400000
−450000
−500000
−550000
−600000
−650000
Energia
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e6
Iteracje

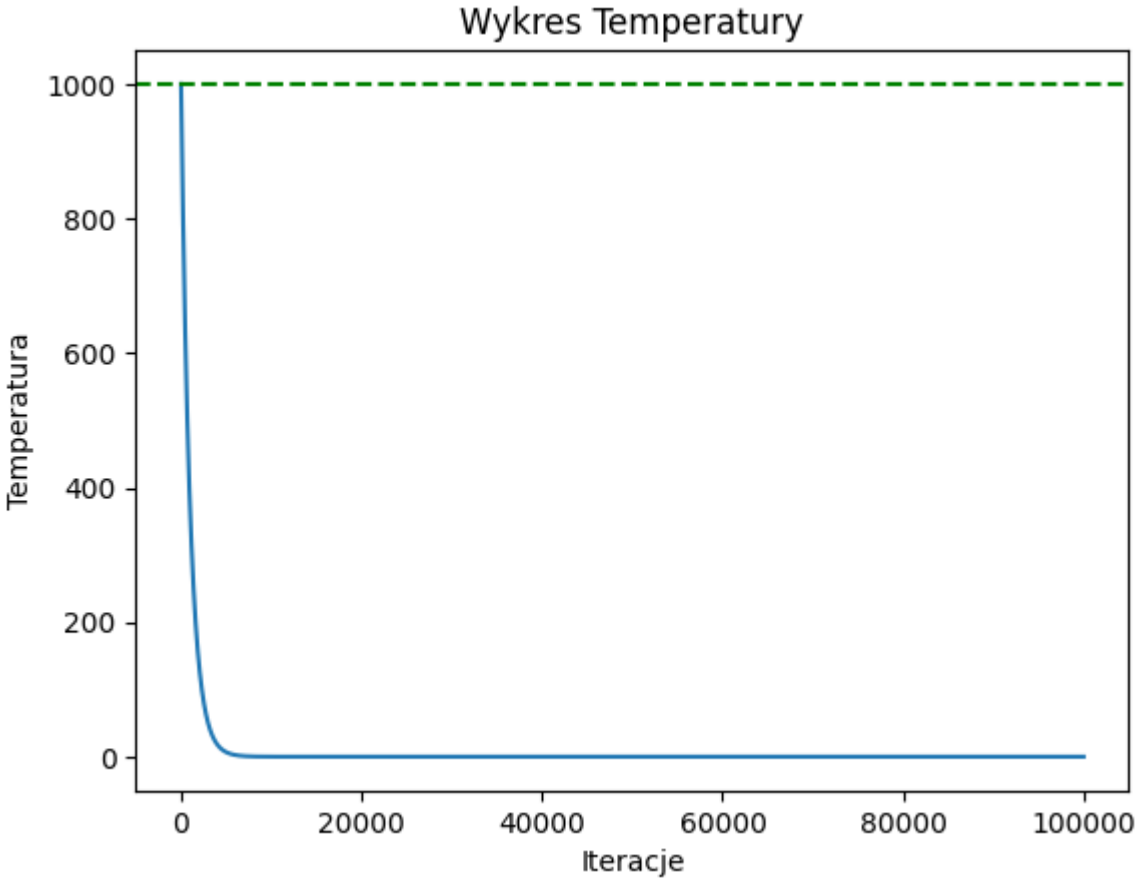

Wykres Temperatury
100
80
60
40
20
0
Temperatura
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e6
Iteracje



Optymalny stan został odnaleziony w czasie 304 sek
Jeśliby pozwolić działać algorytmowi przez dłuższy czas spodziewanym wynikiem byłaby plansza zawierająca jeden zbity czarny zbiór. Poniżej przykład dla innych wartości startowych:

3.2.4.n = 50, δ = 0.4, f = *neighbourhood_energy_combine*, i = 100000, t = 1000, c = 0.999

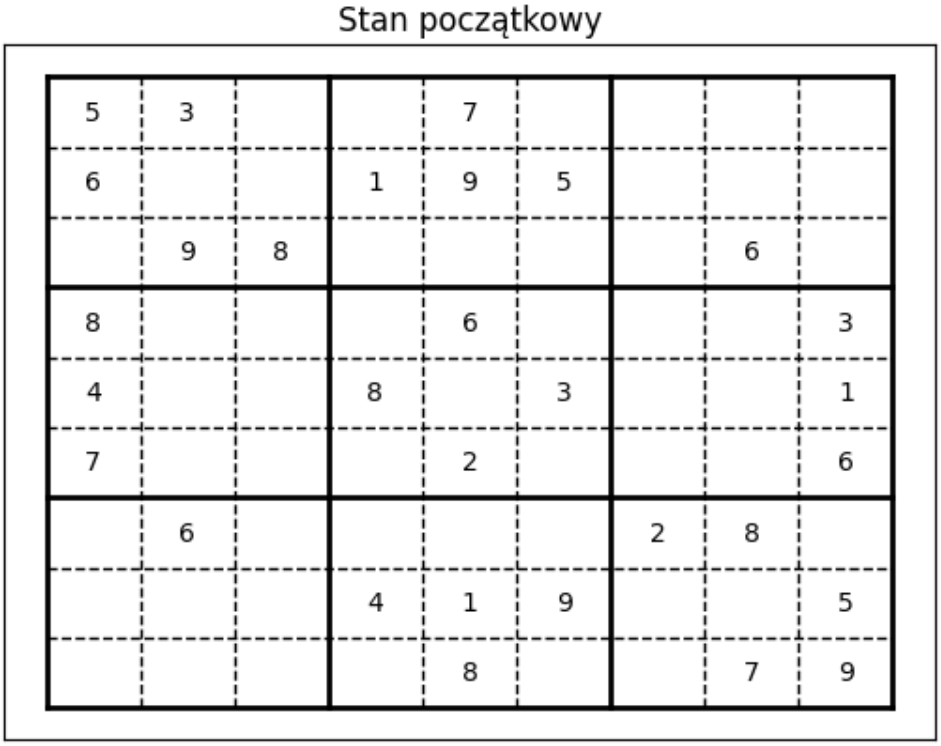

Wykres Enegii
−3000
−4000
−5000
−6000
−7000
Energia
0
20000
40000
60000
80000
100000
Iteracje

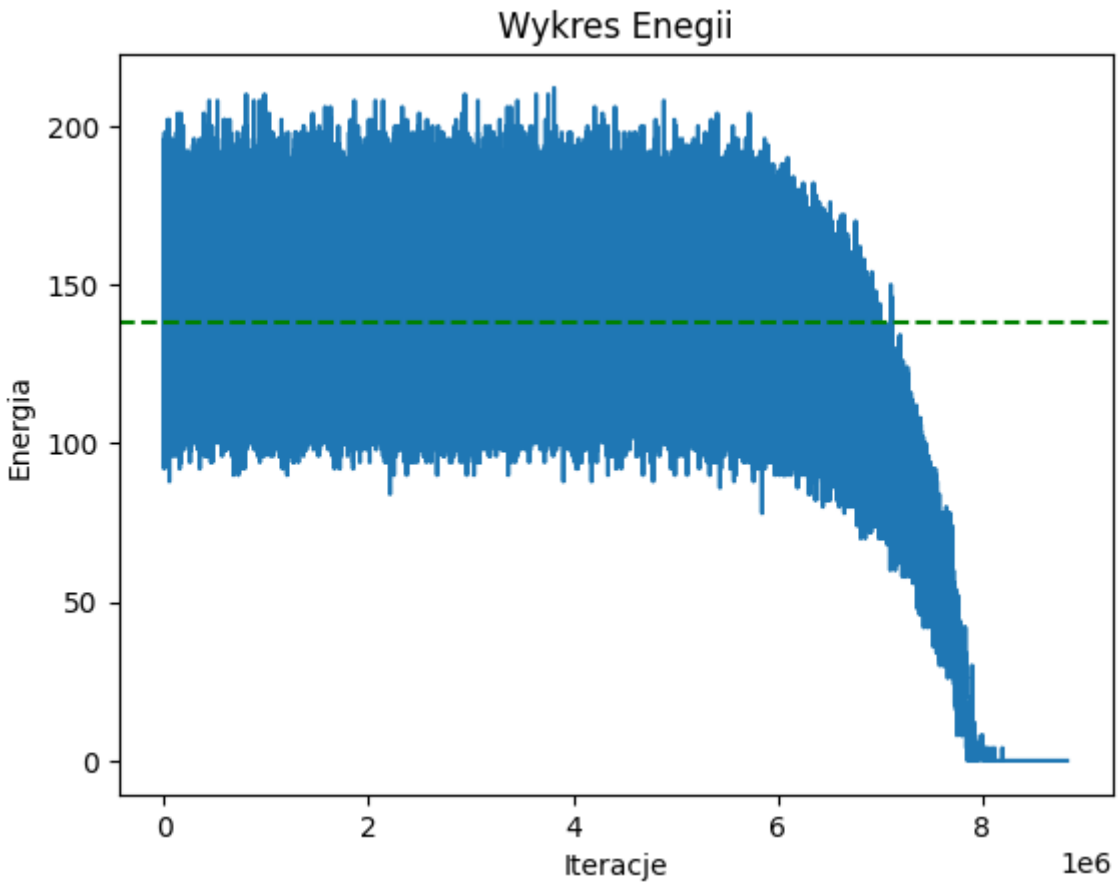

Wykres Temperatury
1000
800
600
400
200
0
Temperatura
0
20000
40000
60000
80000
100000
Iteracje

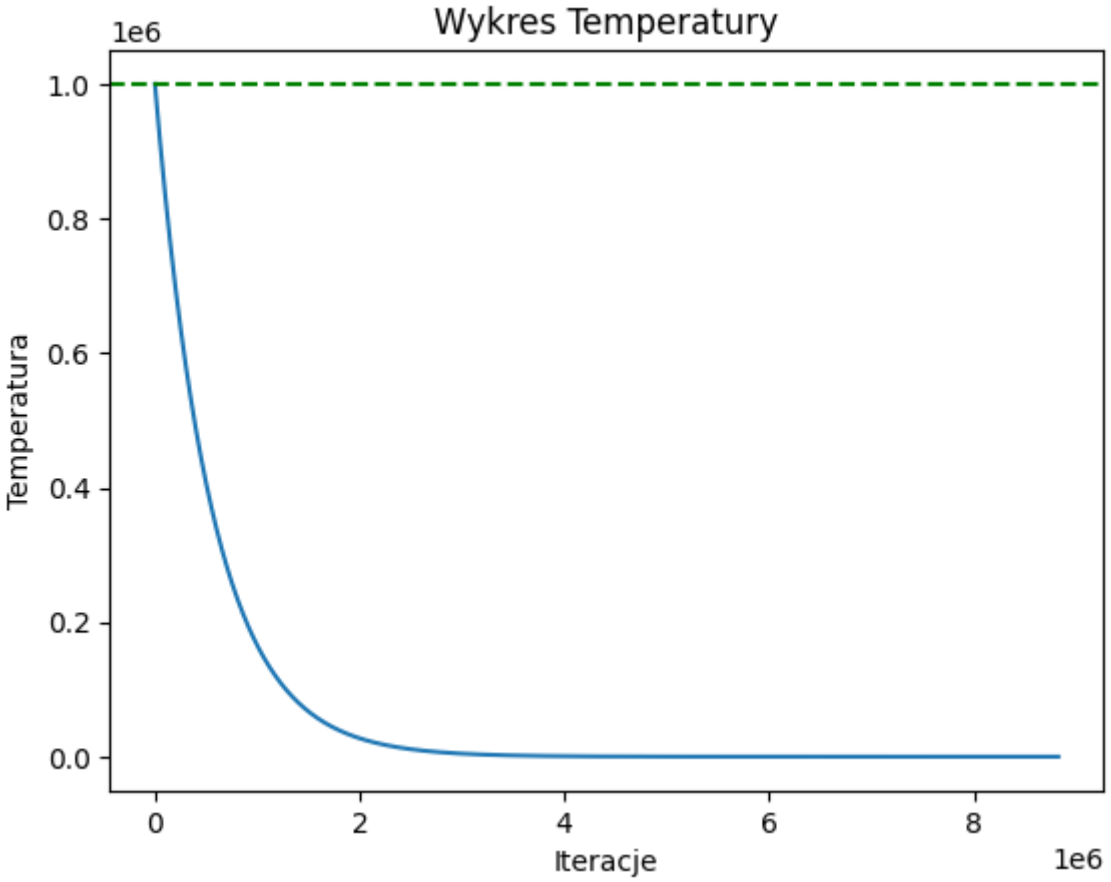
3.3. Plik *zad1_3.ipynb*

Dla obu sudoku zostały zastosowane te same parametry wejściowe:
Maksymalna liczba iteracji: 10000000
Temperatura początkowa: 1000000
Współczynnik schładzania: 0.9999982

3.3.1. **Sudoku z pliku *sudoku1.txt***




Wykres Enegii
Energia
200
150
100
50
0
0
2
4
6
8
Iteracje
1e6


Wykres Temperatury
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Temperatura
0
2
4
6
8
Iteracje
1e6

Stan końcowy

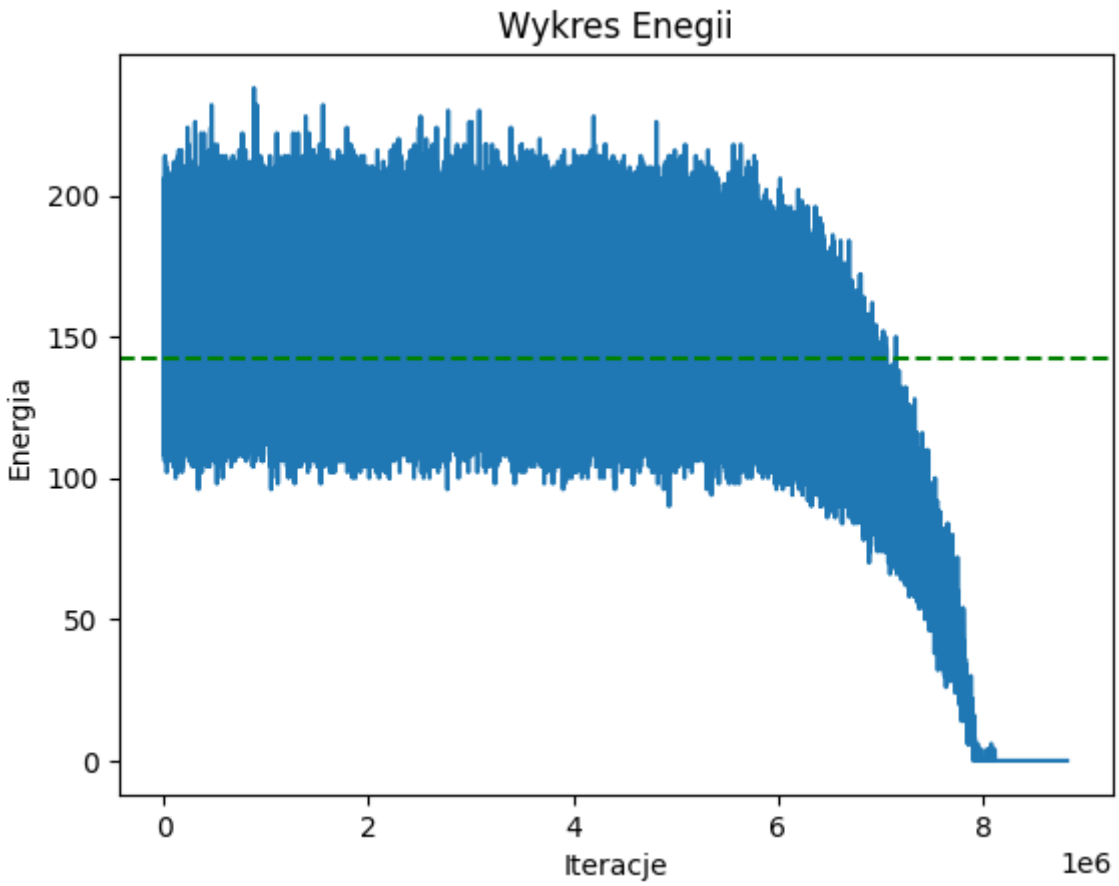
5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

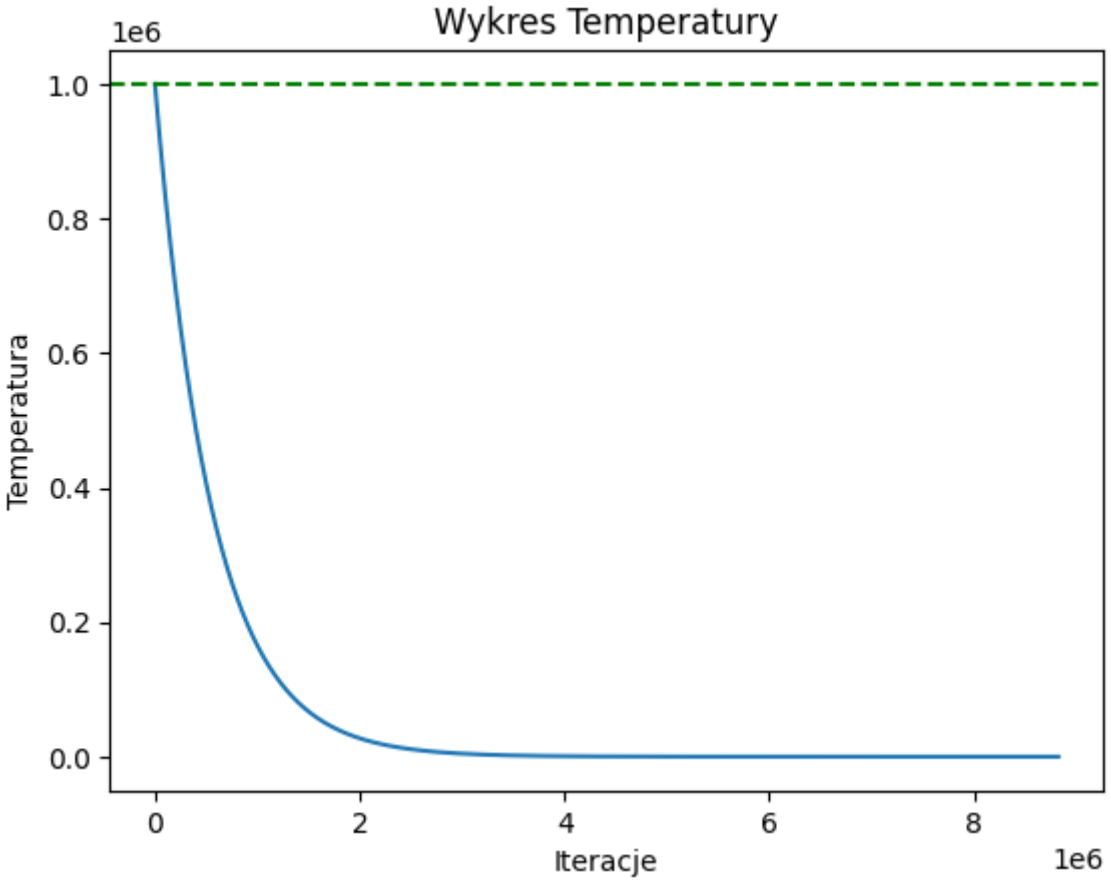
Poprawne rozwiązanie zostało odnalezione w czasie 503 sek

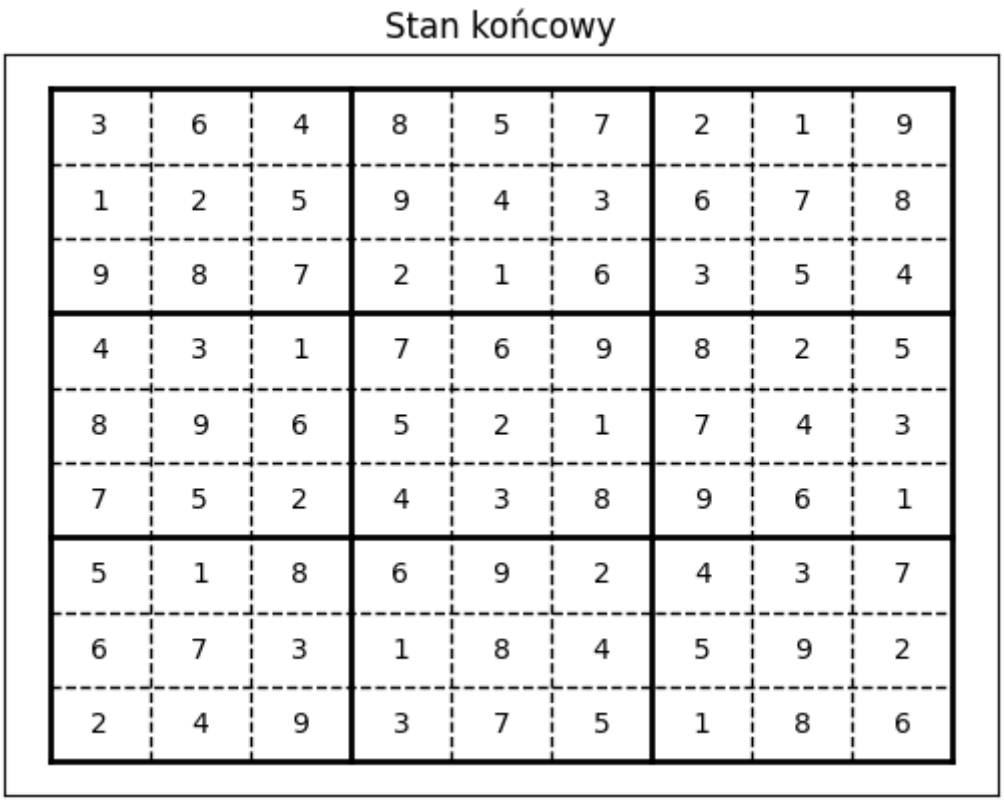
3.3.2. **Sudoku z pliku *sudoku2.txt***

Stan początkowy

		4						
					3		7	8
	8	7						
	3		7					
				2				
					8			1
	1					4		7
6	7							
					5		8	


Wykres Enegii
Energia
200
150
100
50
0
0
2
4
6
8
Iteracje
1e6


Wykres Temperatury
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Temperatura
0
2
4
6
8
Iteracje
1e6



Poprawne rozwiązanie zostało odnalezione w czasie 502 sek

4. Wnioski

4.1. Algorytm symulowanego wyżarzania jest skuteczną metodą rozwiązywania różnorodnych problemów optymalizacyjnych.

4.2. Wszystkie trzy zaimplementowane problemy wykazały tendencję do poprawy wyników wraz z postępem algorytmu.

4.3. Spadek temperatury odzwierciedla proces wyżarzania, podczas gdy spadek wartości funkcji celu świadczy o poprawie rozwiązania.

4.4. Gwałtowne spadki temperatury w punktach 3.2.2, 3.2.3 oraz 3.2.4 wynikają z faktu, że dla podanych funkcji bardziej opłacalne jest wykorzystanie algorytmu naiwnego, polegającego na przyjmowaniu tylko zmian zmniejszających wartość energii. Jest to spowodowane tym, że dla podanych funkcji algorytm bardzo szybko trafia na ścieżkę do minima lokalnego, będącego zwykle globalnym.

4.5. Algorytm jest w stanie znaleźć poprawne rozwiązania dla sudoku które posiadają dokładnie jedno rozwiązanie.

5. Uwagi końcowe

5.1. Implementacje w plikach zad1_1.ipynb, zad1_2.ipynb i zad1_3.ipynb zostały przetestowane i zweryfikowane. Wyniki prezentowane na wykresach odpowiadają oczekiwanym rezultatom algorytmu symulowanego wyżarzania.