

Metody Obliczeniowe w Nauce i Technice

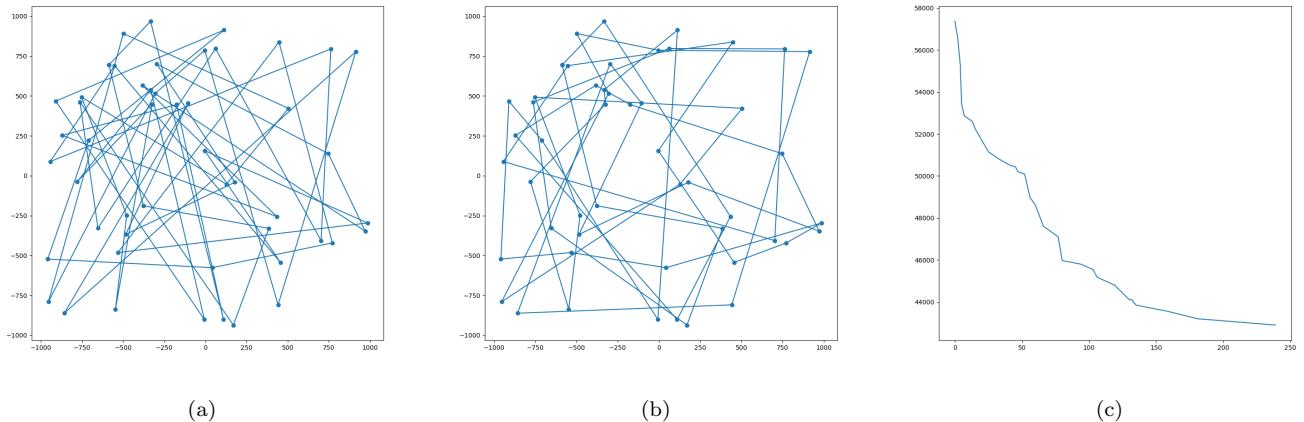
Laboratorium IV

Maciej Trątnowiecki

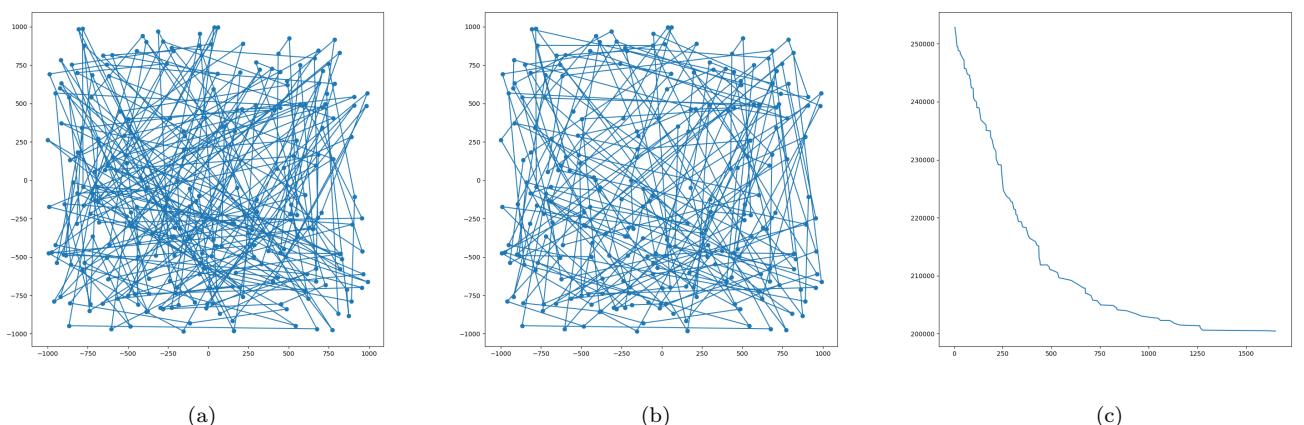
AGH, Semestr Letni, 2020

Problem komiwojażera

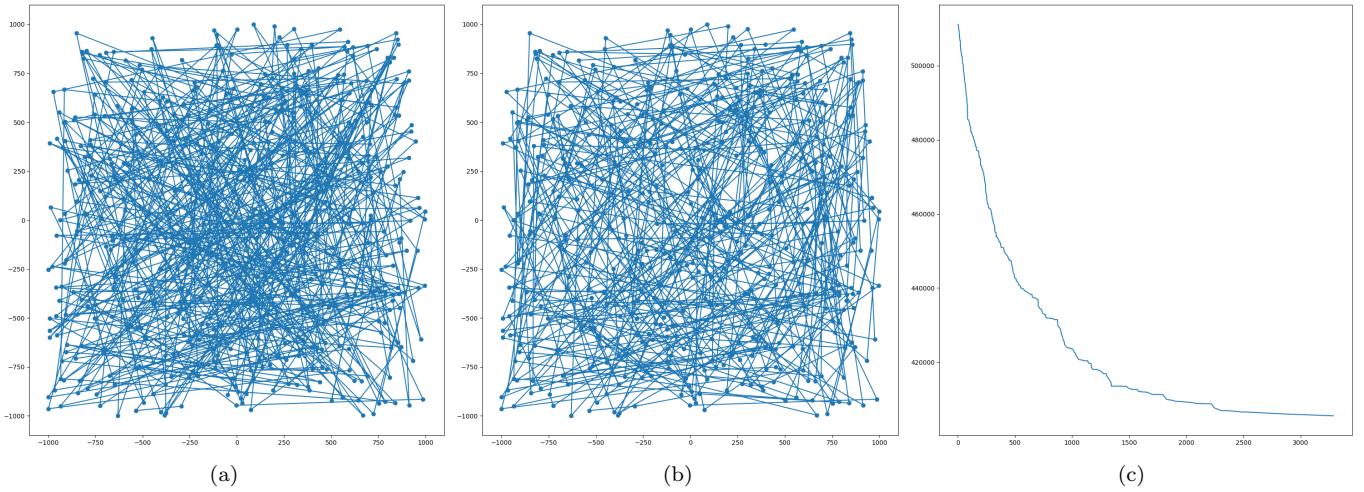
W ramach zadania zaimplementowałem program szukający optymalnej ścieżki w problemie komiwojażera dla losowych punktów na płaszczyźnie dwuwymiarowej. Przetestowałem go dla punktów losowanych z rozkładu jednostajnego, normalnego i podzielonych na 9 odseparowanych grup. Przykładowe wyniki działania programu:



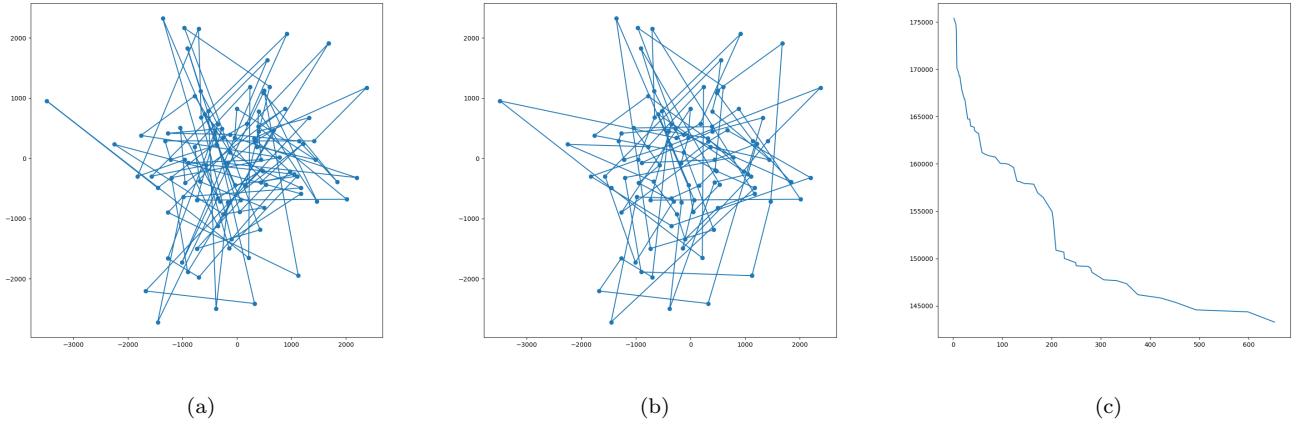
Rysunek 1: Rozkład jednostajny, 100 punktów, temperatura 20, optymalizacja trasy 25.52%



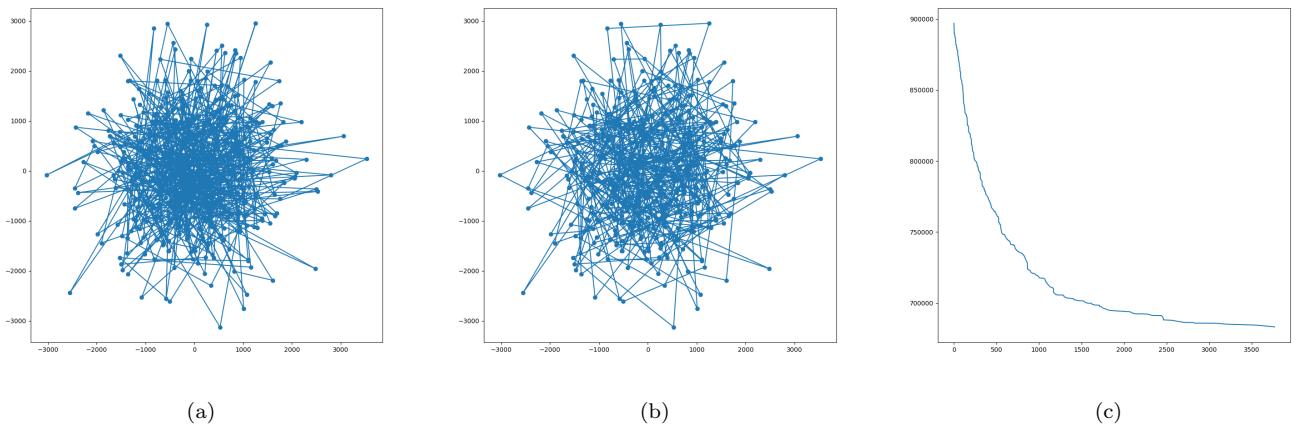
Rysunek 2: Rozkład jednostajny, 500 punktów, temperatura 20, optymalizacja trasy 20.72%



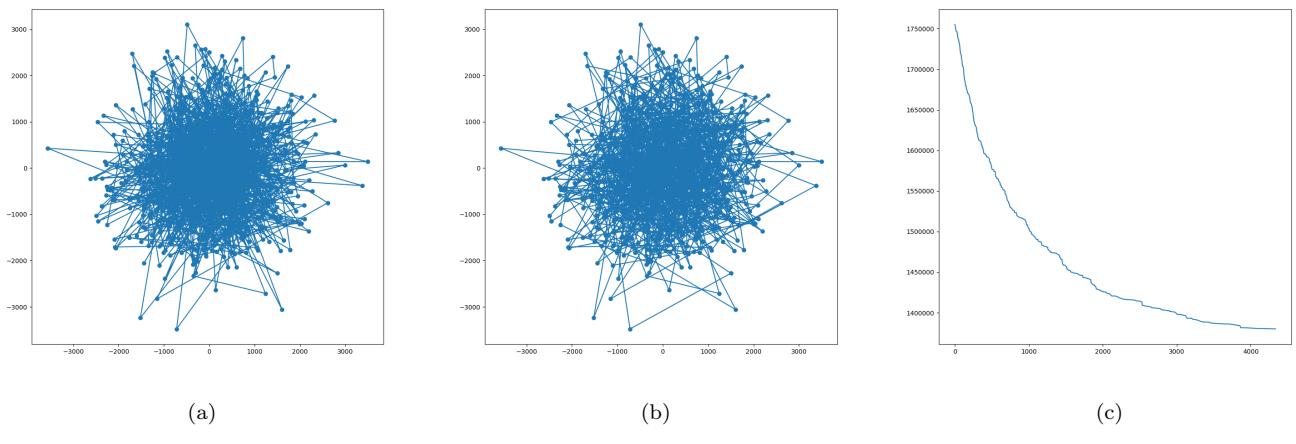
Rysunek 3: Rozkład jednostajny, 1000 punktów, temperatura 31, optymalizacja trasy 20.69%



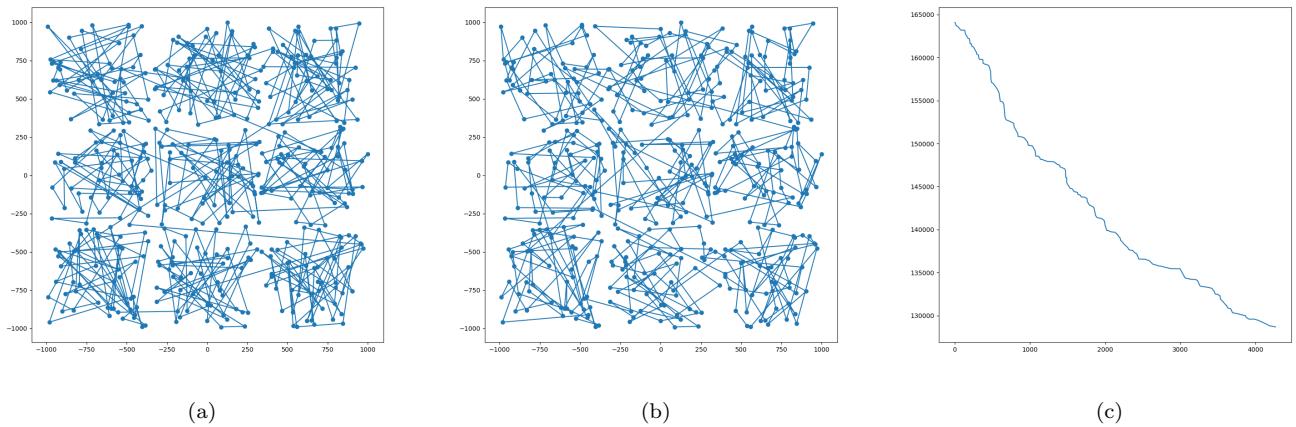
Rysunek 4: Rozkład normalny, 100 punktów, temperatura 20, optymalizacja trasy 18.38%



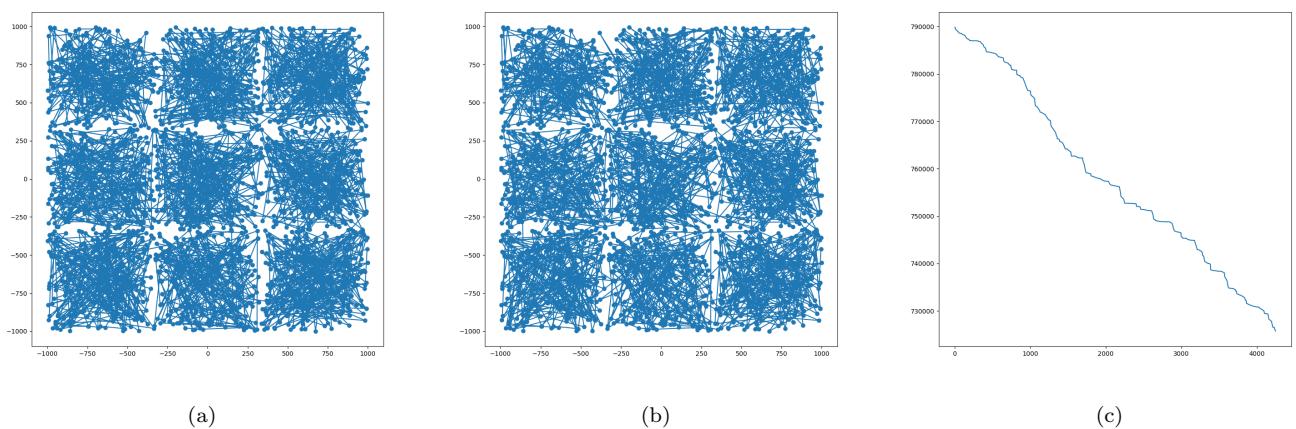
Rysunek 5: Rozkład normalny, 500 punktów, temperatura 20, optymalizacja trasy 24.2%



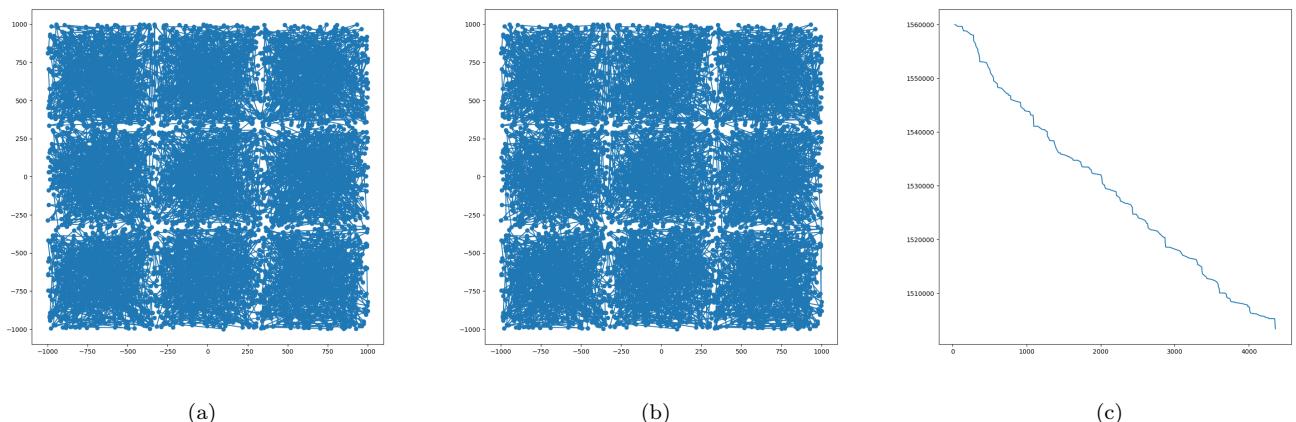
Rysunek 6: Rozkład normalny, 1000 punktów, temperatura 31, optymalizacja trasy 21.41%



Rysunek 7: Rozkład grupowy, 100 punktów, temperatura 20, optymalizacja trasy 21.87%



Rysunek 8: Rozkład grupowy, 500 punktów, temperatura 20, optymalizacja trasy 8.14%



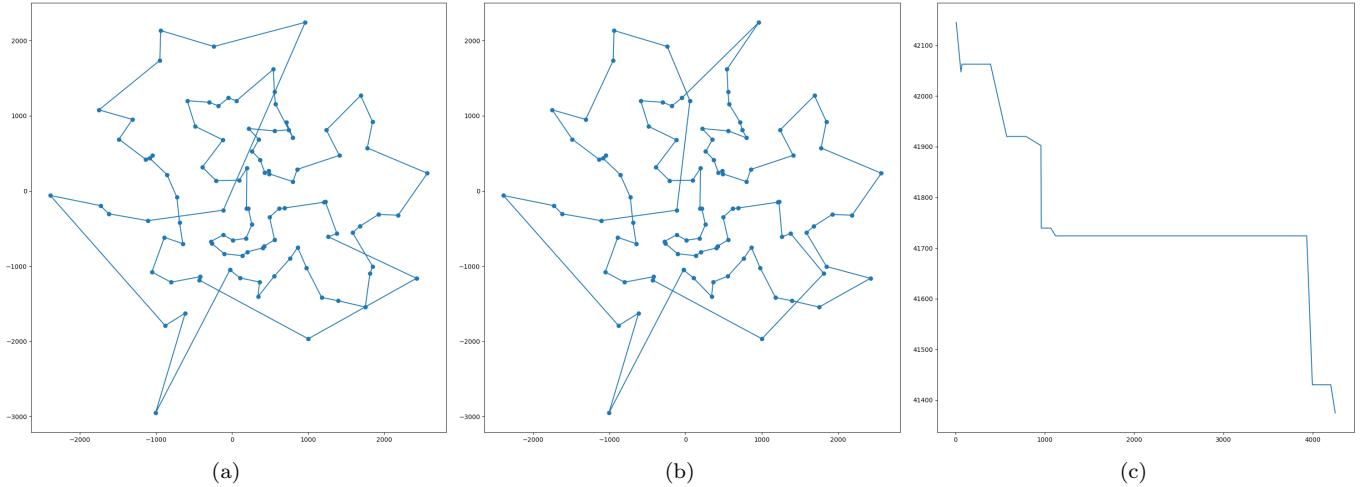
Rysunek 9: Rozkład grupowy, 1000 punktów, temperatura 31, optymalizacja trasy 3.67%

Następnie porównałem obliczoną minimalizację funkcji kosztu w zależności od sposobu generacji stanu sąsiedniego, otrzymując poniższe wyniki (*Arbitrary* - funkcja zamieniająca dwa losowe indeksy, *Consecutive* - funkcja zamieniająca dwa kolejne elementy, *Reverse* - funkcja odwracająca losowy fragment).

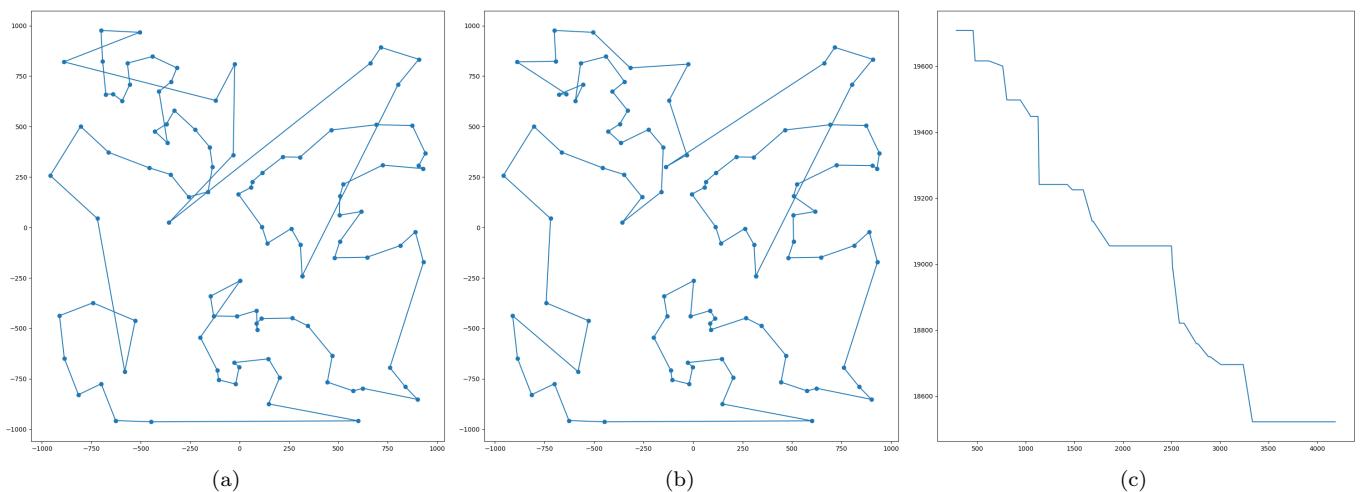
Dane	Arbitrary	Consecutive	Reverse
Rozkład jednostajny, $N = 100$	72.11%	26.45%	78.8%
Rozkład normalny, $N = 100$	61.79%	17.72%	71.34%
Rozkład grupowy, $N = 100$	22.32%	20.51%	21.36%
Rozkład jednostajny, $N = 500$	51.06%	25.2%	60.31%
Rozkład normalny, $N = 500$	47.79%	21.84%	53.97%
Rozkład grupowy, $N = 500$	7.81%	18.91%	5.61%
Rozkład jednostajny, $N = 1000$	44.28%	22.61%	48.16%
Rozkład normalny, $N = 1000$	38.59%	19.67%	43.43%
Rozkład grupowy, $N = 1000$	4.92%	14.8%	3.5%

Tabela 1: Porównanie efektywności działania funkcji.

Następnie przygotowałem funkcję obliczającą zachłanne przybliżenie problemu komiwojażera stosującą heurystykę najbliższego kroku. Dla tak zainicjalizowanych danych, z użyciem funkcji arbitrary swap, przeprowadziłem próbę skuteczności zaimplementowanego algorytmu wyżarzania, otrzymując dużo niższe procentowe optymalizacje.



Rysunek 10: Rozkład normalny, 100 punktów, temperatura 20, optymalizacja trasy 3.6%



Rysunek 11: Rozkład normalny, 100 punktów, temperatura 20, optymalizacja trasy 6.1%

Minimalizacja funkcji energii dla obrazu binarnego

Następnie przygotowałem program generujący losowy, kwadratowy, obraz binarny o zadanym rozmiarze i gęstości czarnych punktów. Przygotowałem różne sąsiedztwa punktu, wyznaczające 4; 8; 16; 16+8 sąsiednich punktów w sposób przedstawiony w poniższej tabeli.

16	16	16	16	16
16	8	8; 4	8	16
16	8; 4		8; 4	16
16	8	8; 4	8	16
16	16	16	16	16

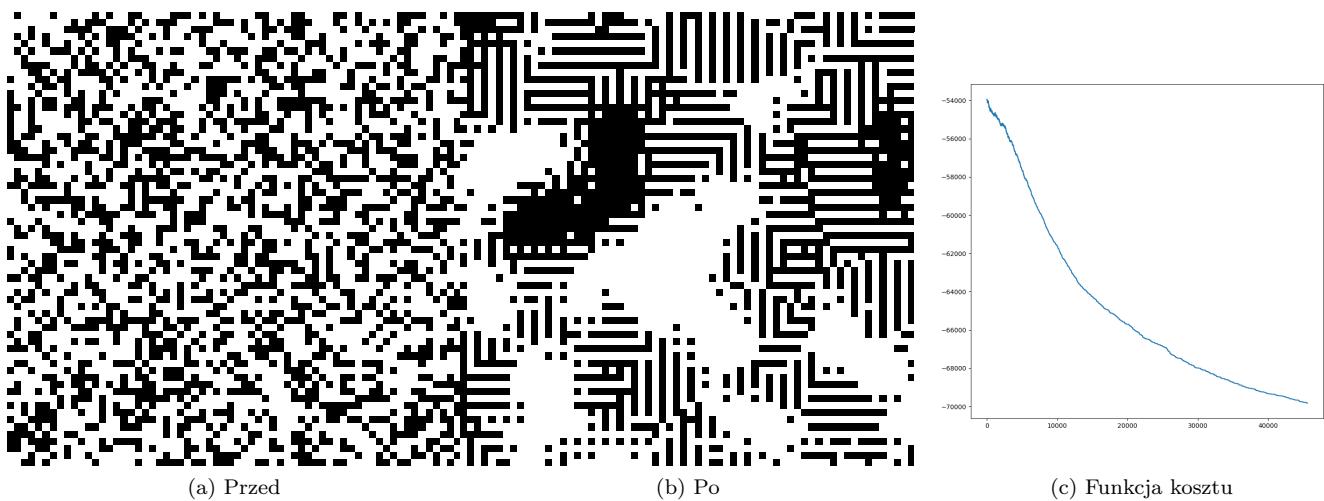
Tabela 2: Graficzna reprezentacja sąsiedztwa.

Przygotowałem też różne funkcje energii punktu, opisane poniżej. Niech d będzie odlegością punktu sąsiadniego od rozpatrywanego w funkcji energii.

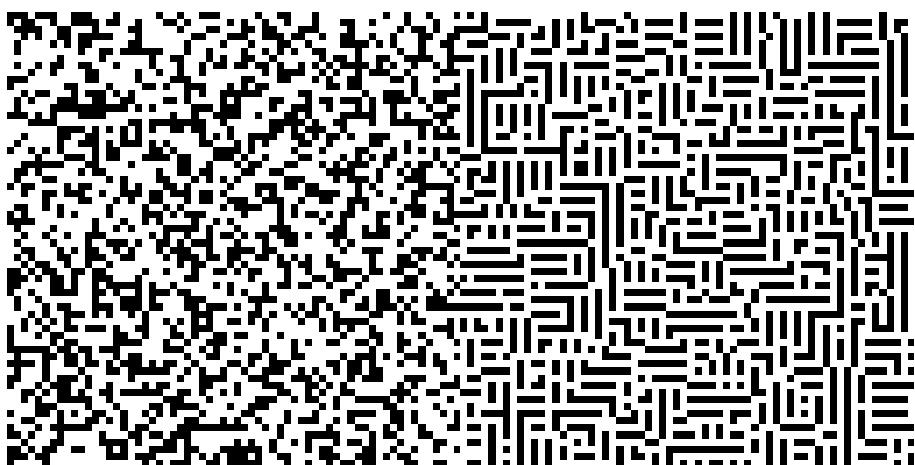
- dist - Jeśli punkt sąsiedni ma tą samą wartość co rozpatrywany, zwraca $-d$ dla punktów o odległości większej niż $\sqrt{2}$, d dla punktów bliższych. W przeciwnym przypadku 0.
 - pushpull - Zwraca d jeśli punkt sąsiedni ma tą samą wartość co rozpatrywany, $-d$ w przeciwnym wypadku.
 - diff - Zwraca $-d$ jeśli punkt sąsiedni ma różną od rozpatrywanego wartość, 0 w przeciwnym przypadku.
 - on - Zwraca $-d$ jeśli zarówno punkt sąsiedni jak i rozpatrywany ma wartość 1, 0 w przeciwnym przypadku.

Tak przygotowany program testowałem z parametrami: Obraz rozmiaru 64x64, ilość kroków algorytmu wyżarzania 10^{12} , temperatura 80, współczynnik $\alpha = 0.9995$. Następny obraz wyznaczany jest poprzez losową zamianę dwóch dowolnych punktów. Funkcja kosztu obrazu zoptymalizowana jest, by w następnym kroku liczyć wartości tylko w tych polach, które uległy zmianie.

Przykładowe wyniki działania programu:

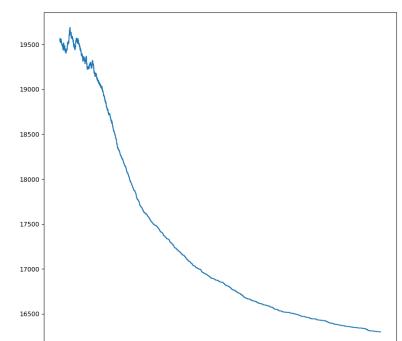


Rysunek 12: Sąsiedztwo: 16+8, funkcja energii: dist.



(a) Przed

(b) Po



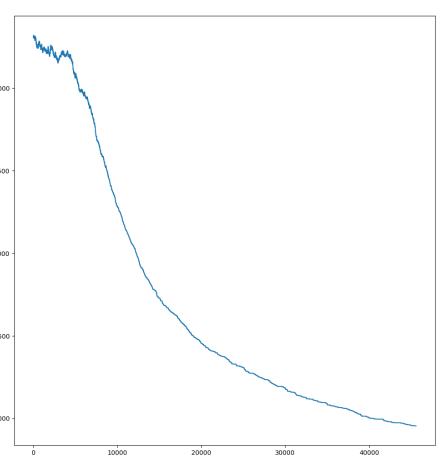
(c) Funkcja kosztu

Rysunek 13: Sąsiedztwo: 8, funkcja energii: dist.



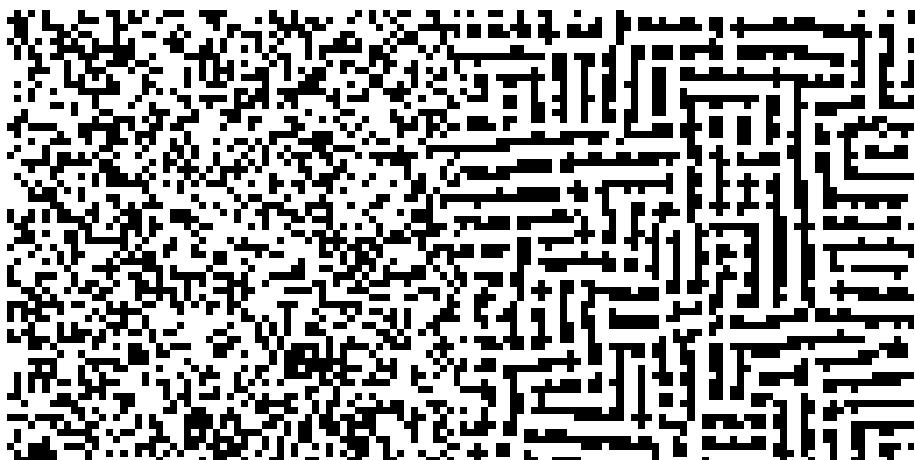
(a) Przed

(b) Po



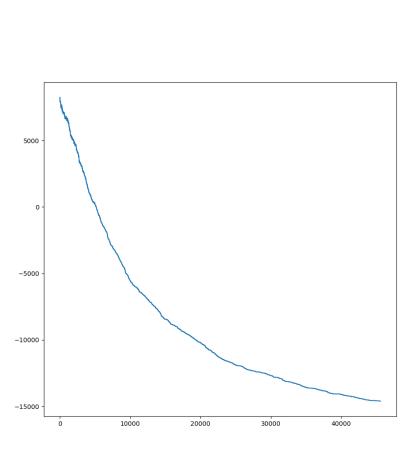
(c) Funkcja kosztu

Rysunek 14: Sąsiedztwo: 4, funkcja energii: dist.



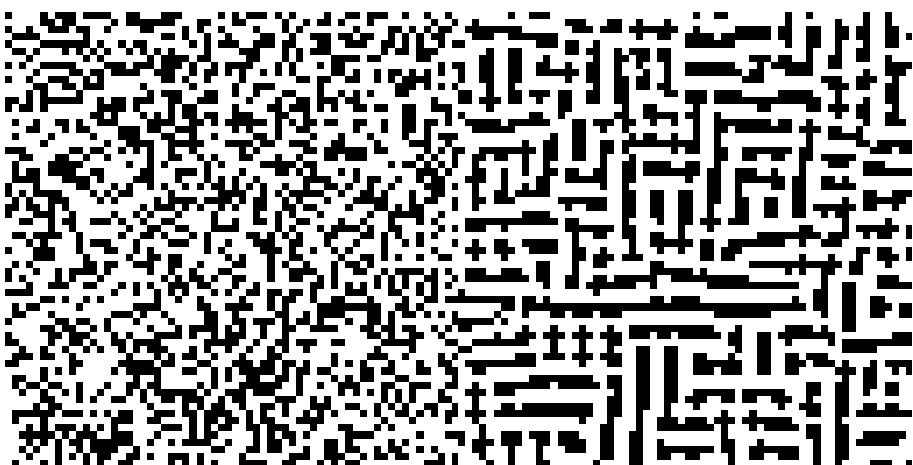
(a) Przed

(b) Po



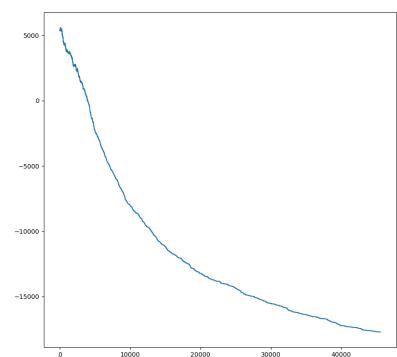
(c) Funkcja kosztu

Rysunek 15: Sąsiedztwo: 16+8, funkcja energii: pushpull.



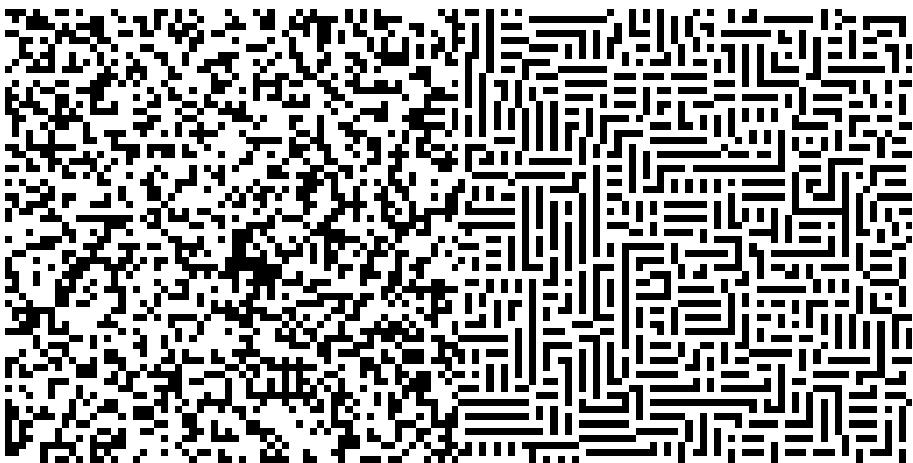
(a) Przed

(b) Po



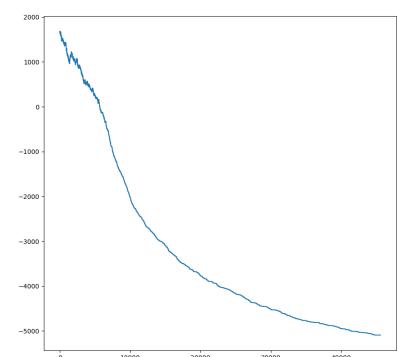
(c) Funkcja kosztu

Rysunek 16: Sąsiedztwo: 16, funkcja energii: pushpull.



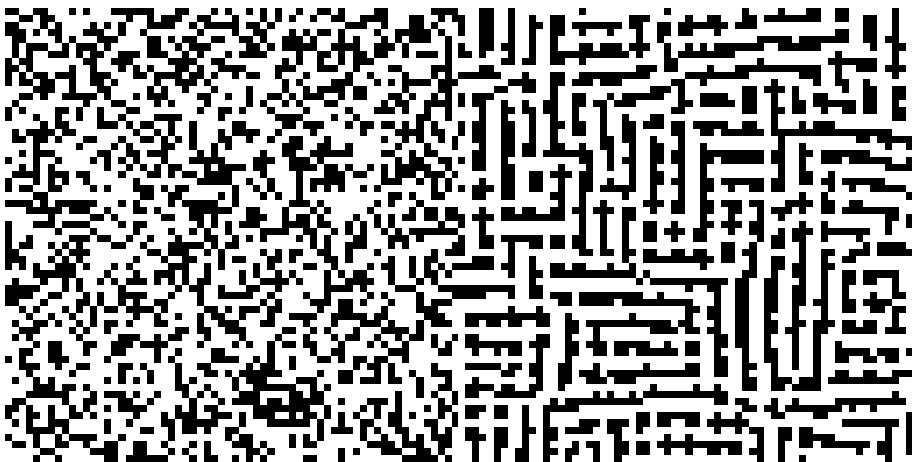
(a) Przed

(b) Po



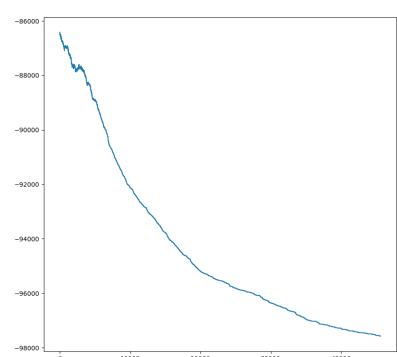
(c) Funkcja kosztu

Rysunek 17: Sąsiedztwo: 8, funkcja energii: pushpull.



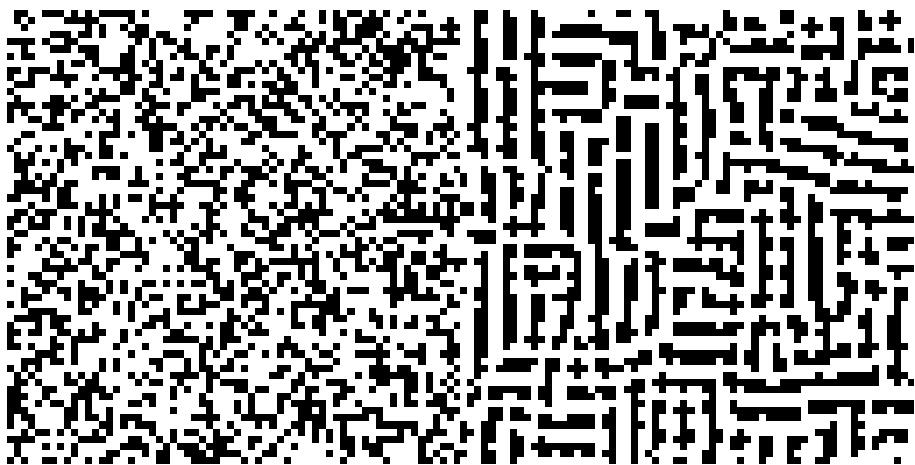
(a) Przed

(b) Po



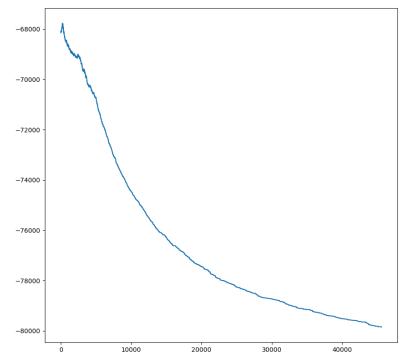
(c) Funkcja kosztu

Rysunek 18: Sąsiedztwo: 16+8, funkcja energii: diff.



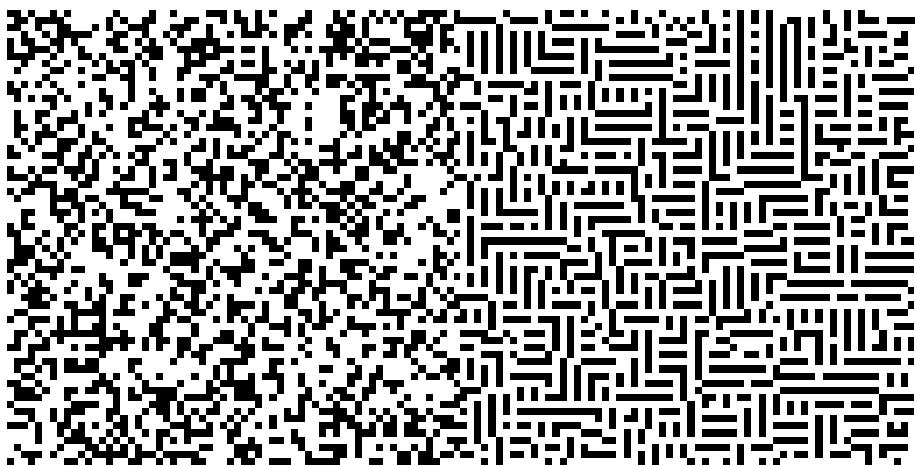
(a) Przed

(b) Po



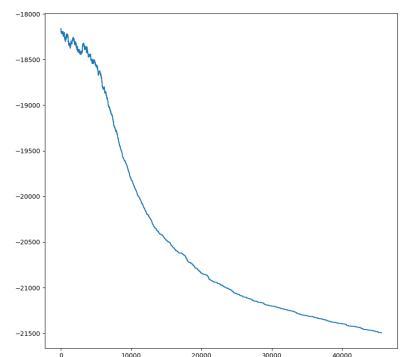
(c) Funkcja kosztu

Rysunek 19: Sąsiedztwo: 16, funkcja energii: diff.



(a) Przed

(b) Po



(c) Funkcja kosztu

Rysunek 20: Sąsiedztwo: 8, funkcja energii: diff.

Sudoku

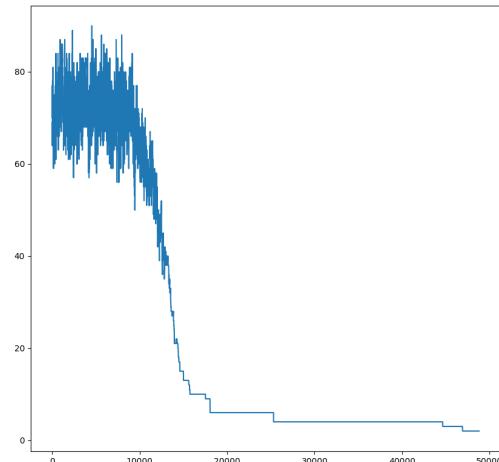
Przygotowałem program poszukujący rozwiązań dla planszy sudoku metodą symulowanego wyżarzania. Jako funkcję kosztu przyjąłem sumę powtórzeń cyfr w wierszach i kolumnach planszy, oraz w blokach o rozmiarze 3x3. Przy wczytywaniu planszy z pliku tekstowego, oznaczam puste pola jako modyfikowalne. Uzupełniam je także brakującymi w całej planszy cyframi. Stan następny wyznaczam zamieniając dowolne dwie modyfikowalne pozycje.

Program testowałem z użyciem parametrów: temperatura 400, współczynnik $\alpha = 0.9995$, liczba kroków 10^{12} .

Przykłady działania programu.

Problem	Solved board
2 x x 6 x 7 5 x x	2 3 4 6 9 7 5 8 1
x x x x x x x 9 6	1 8 5 3 2 4 7 9 6
6 x 7 x x 1 3 x x	6 9 7 8 5 1 3 4 2
x 5 x 7 3 2 x x x	4 5 1 7 3 2 8 6 9
x 7 x x x x x 2 x	9 7 8 4 6 5 1 2 3
x x x 1 8 9 x 7 x	3 6 2 1 8 9 4 7 5
x x 3 5 x x 6 x 4	9 2 3 5 7 8 6 1 4
8 4 x x x x x x x	8 4 6 9 1 3 2 5 7
x x 5 2 x 6 x x 8	7 1 5 2 4 6 9 3 8
Cost: 2	

(a)

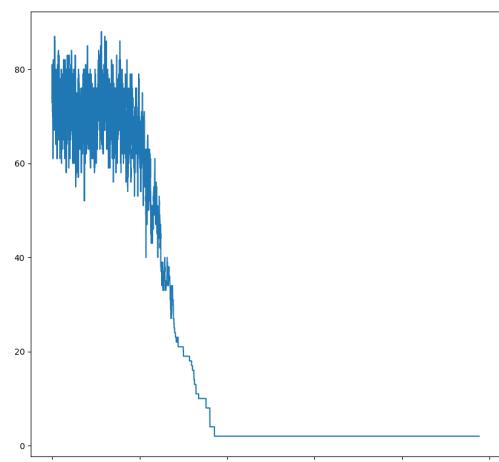


(b)

Rysunek 21: Przykład 1.

Problem	Solved board
x x 2 x x x 5 x x	9 7 2 4 6 3 5 8 1
x 1 x 7 x 5 x 2 x 6	1 8 7 2 5 9 2 3
4 x x x 9 x x x 7	4 5 3 8 9 1 6 4 7
x 4 9 x x x 7 3 x 5	4 9 2 1 8 7 3 6
8 x 1 x 3 x 4 x 9 8	2 1 6 3 7 4 5 9
x 3 6 x x x 2 1 x 7	3 6 5 4 9 2 1 8
2 x x x 8 x x x 4 2	9 5 1 8 6 3 7 4
x 8 x 9 x 2 x 6 x 3	8 4 9 7 2 1 6 5
x x 7 x x x 8 x x 1	6 7 3 5 4 8 9 2
Cost: 2	

(a)

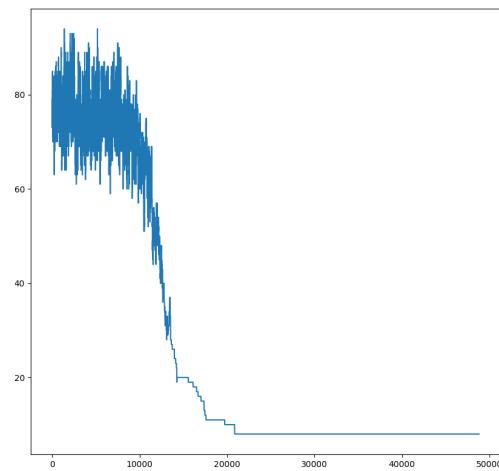


(b)

Rysunek 22: Przykład 2.

Problem	Solved board
x 4 x x x x x x x	3 4 9 5 7 1 6 8 9
8 x 6 x x x x x x	7 8 1 6 2 4 3 2 5 7
5 x x x x x 1 x x	5 2 7 8 6 9 1 4 3
x x 5 1 9 x x 6 x	9 3 5 1 9 2 8 6 4
x x 4 x 5 x x x x	1 8 4 6 5 7 3 7 2
x x x 3 x x x 9 1	7 6 2 3 8 4 5 9 1
x x x x x 2 4 3 6	4 7 1 9 5 2 4 3 6
x 9 x x x x x x x	2 9 8 4 3 6 7 1 5
x x x x x 8 x 2 x	6 5 3 7 1 8 9 2 8
Cost: 8	

(a)

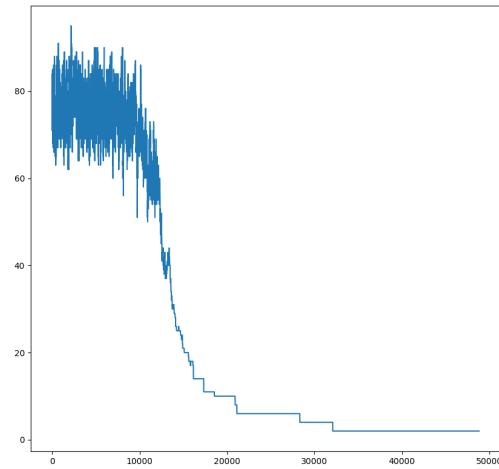


(b)

Rysunek 23: Przykład 3.

Problem	Solved board
1 x x x x 7 x 9 x	1 2 4 8 5 7 6 9 3
x 3 x x 2 x x x 8 5	3 6 4 2 9 1 7 8
x x 9 6 x x 5 x x	7 8 9 6 3 1 5 2 4
x x 5 3 x x 9 x x	2 7 5 3 1 5 9 4 6
x 1 x x 8 x x x 2 4	1 3 9 8 6 7 5 2
6 x x x x 4 x x x 6 9	8 2 7 4 8 3 1
3 x x x x x x 1 x 3	5 2 7 6 8 4 1 9
x 4 x x x x x x 7 8	4 1 5 9 3 2 6 7
x x 7 x x x 3 x x 9	6 7 1 4 2 3 8 5
Cost: 2	

(a)

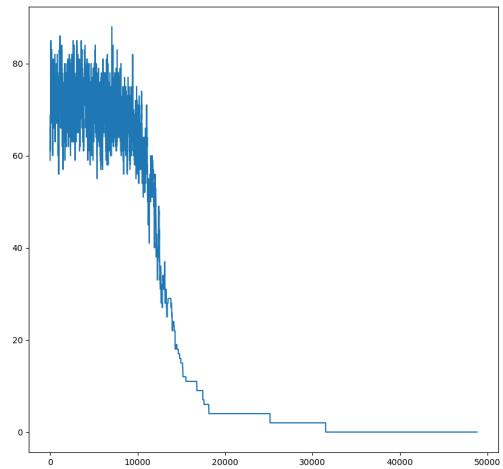


(b)

Rysunek 24: Przykład 4.

Problem	Solved board
x 4 x x x x 1 7 9	8 4 5 6 3 2 1 7 9
x x 2 x x 8 x 5 4	7 3 2 9 1 8 6 5 4
x x 6 x x 5 x x 8	1 9 6 7 4 5 3 2 8
x 8 x x 7 x 9 1 x	6 8 3 5 7 4 9 1 2
x 5 x x 9 x x 3 x	4 5 7 2 9 1 8 3 6
x 1 9 x 6 x x 4 x	2 1 9 8 6 3 5 4 7
3 x x 4 x x 7 x x	3 6 1 4 2 9 7 8 5
5 7 x 1 x x 2 x x	5 7 4 1 8 6 2 9 3
9 2 8 x x x x 6 x	9 2 8 3 5 7 4 6 1
Cost: 0	

(a)

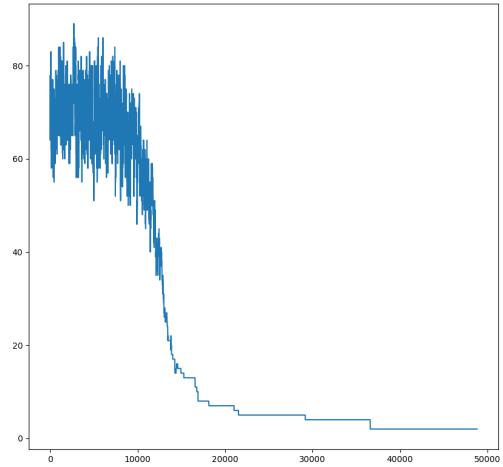


(b)

Rysunek 25: Przykład 5.

Problem	Solved board
x x x x x x x x x	7 4 9 5 6 1 8 2 3 7
7 2 x 3 x 9 x x 1	7 2 6 3 4 9 5 8 1
x x 8 7 x 5 x 6 x	3 1 8 7 2 5 4 6 9
5 x 2 8 9 x x x x	5 7 2 8 9 4 3 1 6
x 4 x 5 x 1 x 9 x	6 4 3 5 7 1 8 9 2
x x x x 6 3 7 x 5 1	8 9 2 6 3 7 4 5
x 3 x 9 x 6 1 x x 8	3 7 9 5 6 1 2 4
2 x x 1 x 7 x 5 3 2	6 4 1 8 7 9 5 3
9 x x x x x x x x x	9 5 1 4 3 2 6 7 8
Cost: 0	

(a)



(b)

Rysunek 26: Przykład 6.

Zauważylem, że program nie zwraca poprawnego rozwiązania planszy sudoku w każdym przypadku. Koszt końcowej planszy waha się także pomiędzy uruchomieniami programu.