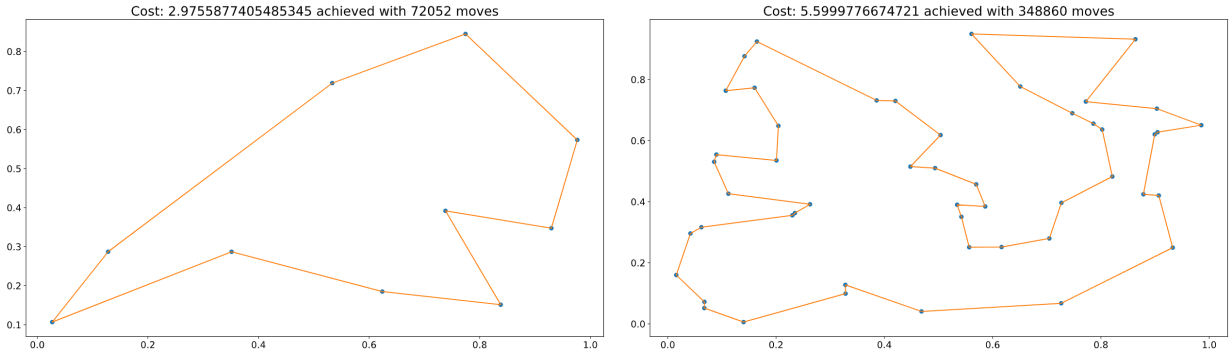
Metody Obliczeniowe w Nauce i Technice
Laboratorium 4
Symulowane wyżarzanie

Patryk Wojtyczek

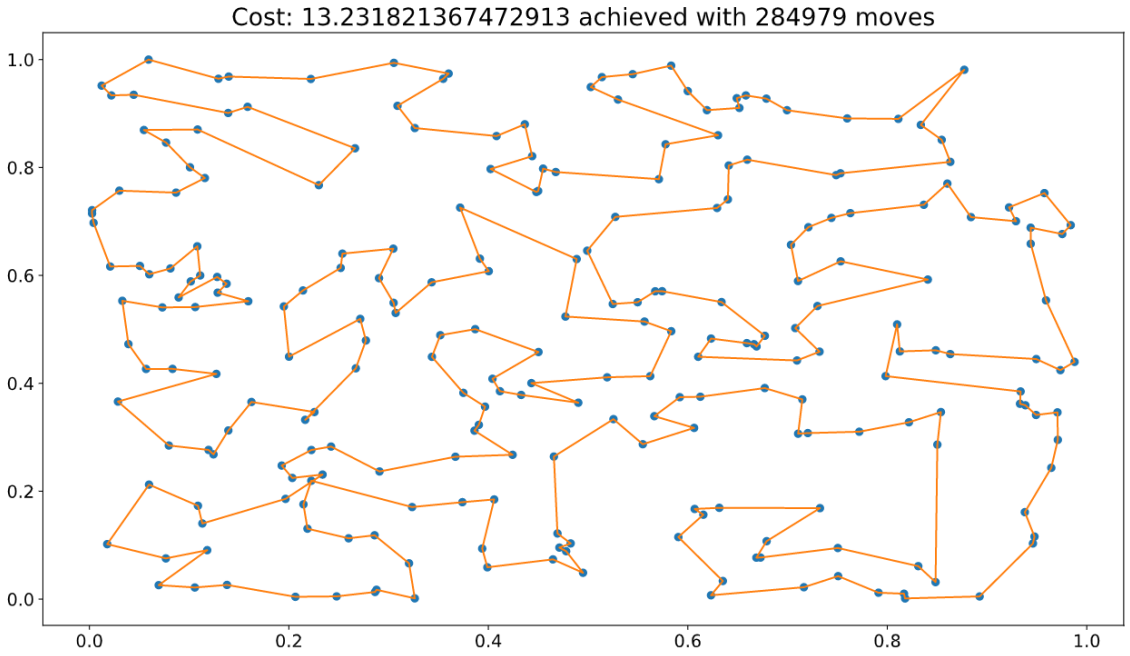
Zadanie 1 - TSP

Wygeneruj chmurę n losowych punktów w 2D, a następnie zastosuj algorytm symulowanego wyżarzania do przybliżonego rozwiązania problemu komiwojażera dla tych punktów.

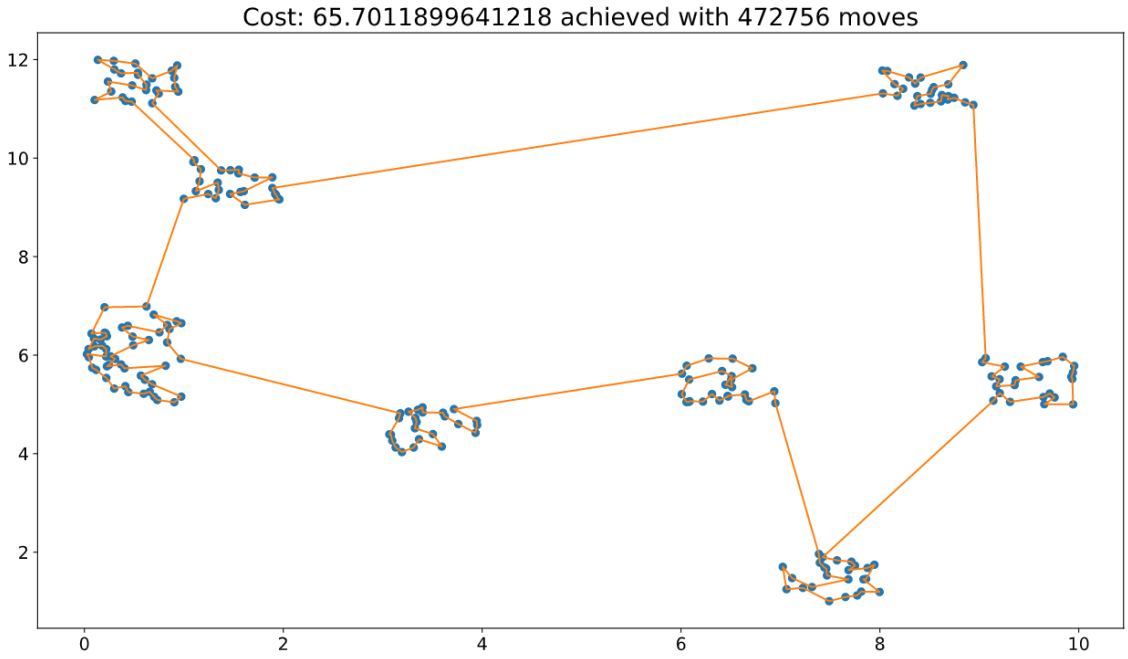
Wizualizacja przykładowych rozwiązań



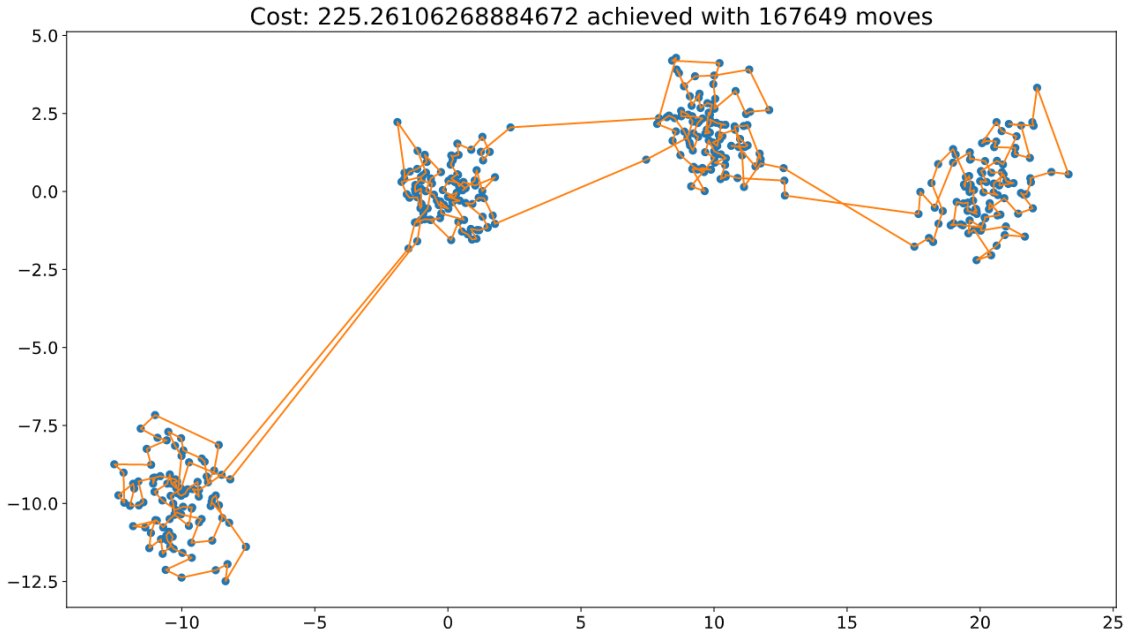
Rysunek 1: Losowa chmura punktów 10 i 50



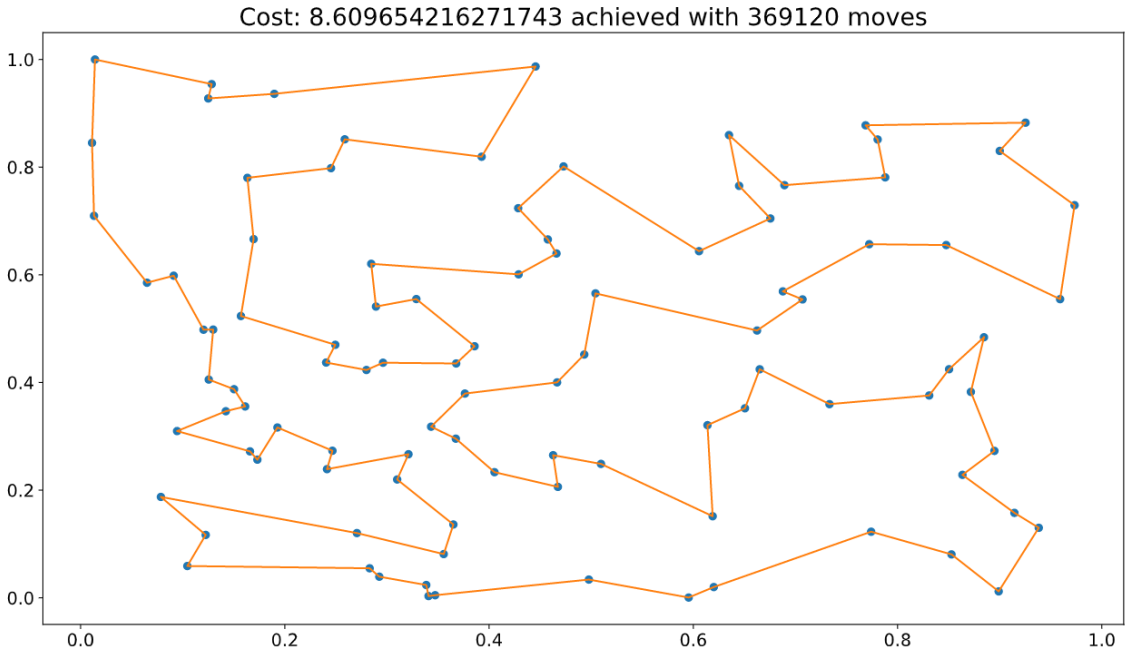
Rysunek 2: Losowa chmura 250 punktów



Rysunek 3: 8 rozłącznych zbiorów punktów



Rysunek 4: Punkty samplowane z dwuwymiarowego rozkładu normalnego, 4 zbiory parametrów

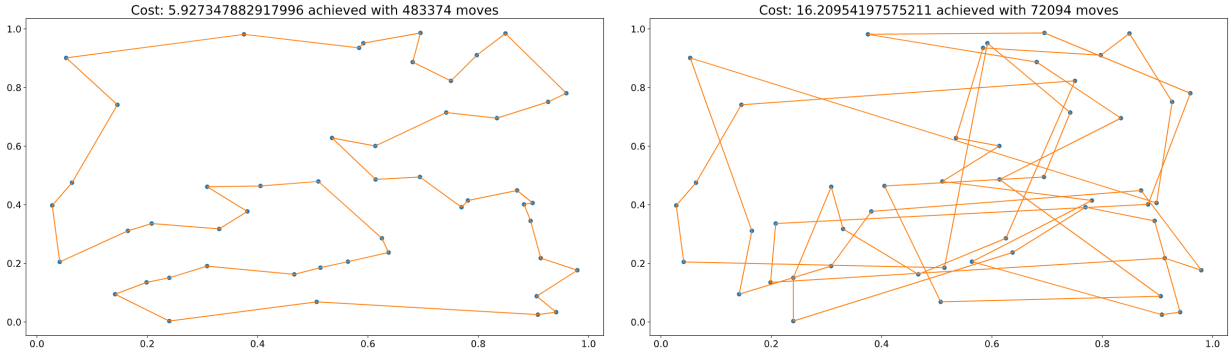


Rysunek 5: 100 punktów z rozkładu jednostajnego

Wpływ funkcji generującej sąsiadów na zbieżność

Używając funkcji arbitrary_swap nasz algorytm jest podatny na znajdowanie rozwiązań zarówno bardzo dobrych jak i tych bardzo złych - w miarę jak system się ochładza tylko dobre rozwiązania są akceptowane zatem to podejście zapewnia nam dużą szansę na znalezienie rozwiązania bliskiego celu - duża przestrzeń możliwych rozwiązań zostanie przeszukana.

Funkcja consecutive_swap jest znacznie bardziej ostrożna - nie wprowadza dużych zmian do znalezionego rozwiązania co jest dobrym podejściem gdy zbliżamy się do globalnego minmum, jednak na początku zależy nam na tym aby przeszukać jak najwięcej możliwych rozwiązań. Funkcja ta ma tendencję do znalezienia minimum lokalnego i utknięciu w nim ze względu na małą ilość wprowadzanych zmian. Do rozwiązania zbiega jednak znacznie szybciej niż podejście wyżej bo zatrzymuje się na znalezionym minimum. Aby uzyskać lepszy rezultat moglibyśmy zwiększyć początkową temperaturę co w pewnym stopniu zwiększyłoby ilość przeszukiwanych rozwiązań.



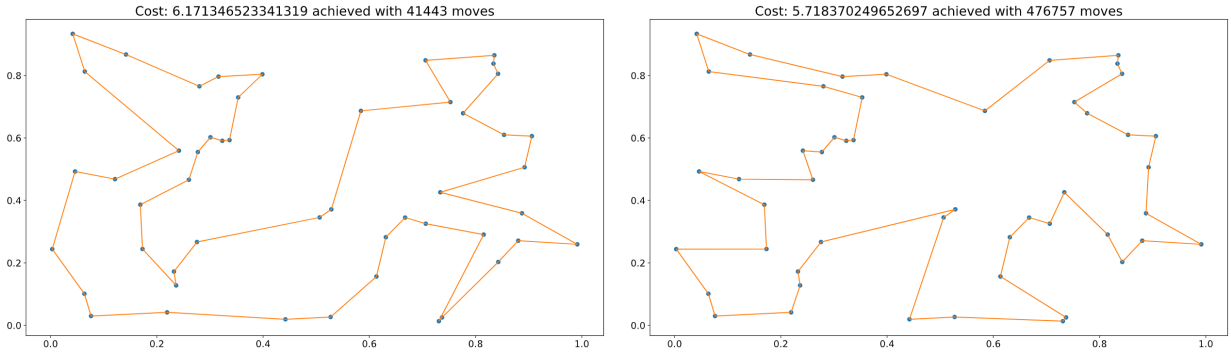
Rysunek 6: Rozwiązania znalezione dla tego samego zbioru punktów używając kolejno arbitrarySwap i consecutiveSwap

Wpływ funkcji temperatury na zbieżność

Jako funkcję temperatury wykorzystałem ‘exponential decay‘ z różnymmi współczynnikami:

- 0.9 - exponential_decay_v9 (domyślna)
- 0.5 - exponential_decay_v5

Rysunki poniżej pokazują, że rozwiązanie znalezione przez v5 jest niewiele gorsze od tego znalezionego przez v9 natomiast ilość kroków (moves) była ponad 10 razy większa. Bierze się to z faktu, że system używający v5 znacznie szybciej się ochładza i nie przeszukuje aż tak dużej liczby rozwiązań co v9 natomiast dalej znajduje rozsądnie wyglądające rozwiązanie.



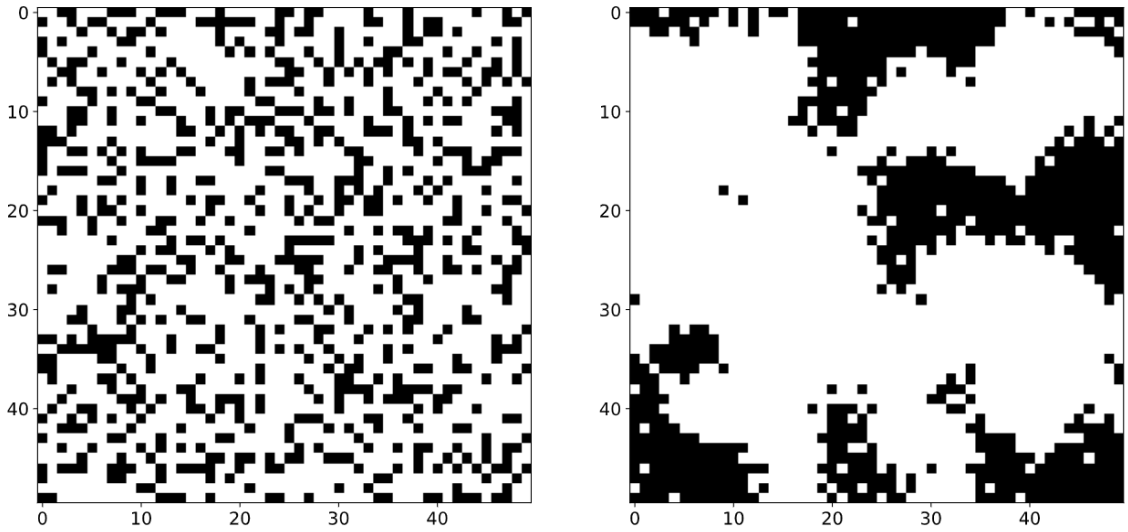
Rysunek 7: Po lewej v5, po prawej v9

Animacja działania algorytmu

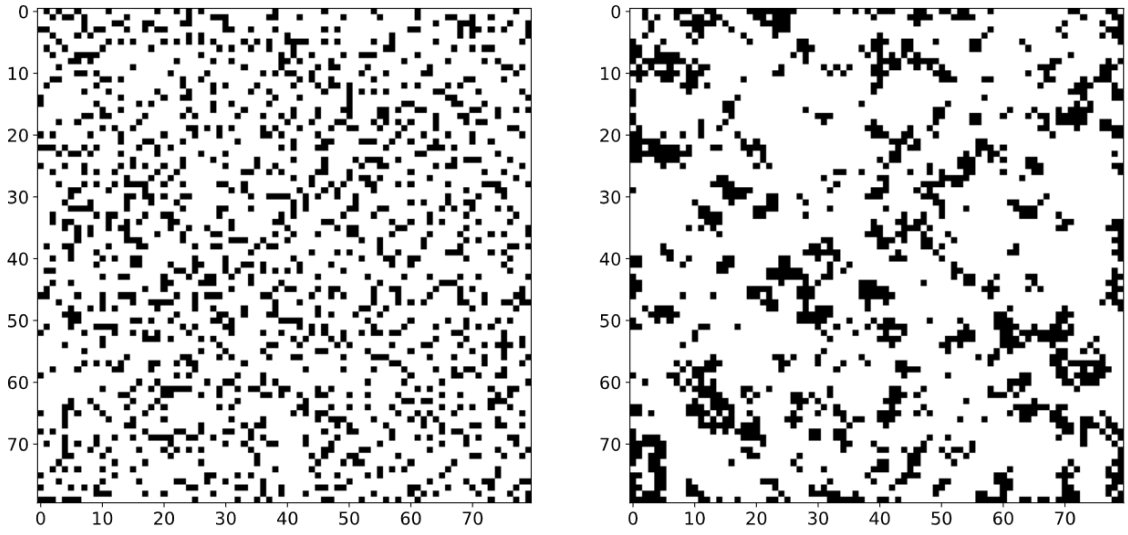
Aby uruchomić animację do funkcji solve wystarczy podać dodatowy argument: ‘animate = True‘. Animacje utworzone z v9 wyglądają znacznie lepiej natomiast szukanie rozwiązania może trwać długo, aby szybko uzyskać rozwiązanie możemy zmienić domyślną funkcję temperaturu poprzez ‘tempF = exponential_decay_v5’. Do uruchomienia animacji jest potrzebny pakiet ffmpeg, który można zainstalować poprzez polecenie ‘conda install -c menpo ffmpeg‘. Na końcu notebooka są przykłady, które trzeba ręcznie uruchomić gdyż zajmują dużo pamieci.

Zadanie 2 - Obraz binarny

Przykładowe obrazy



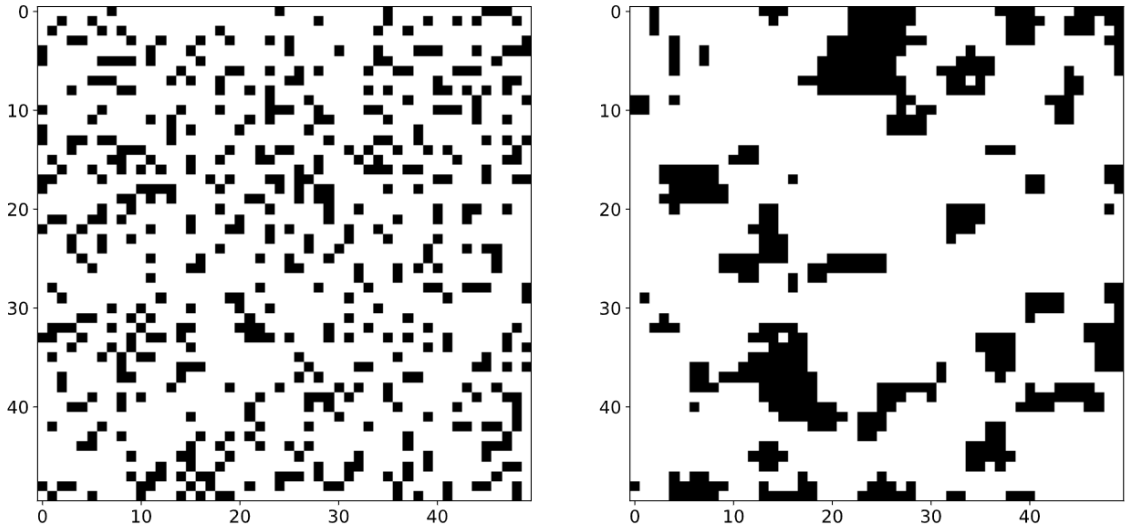
Rysunek 8: Wynik zastosowania symulowanego wyżarzana na losowym obrazie binarnym o gęstości $\delta = 0.3$



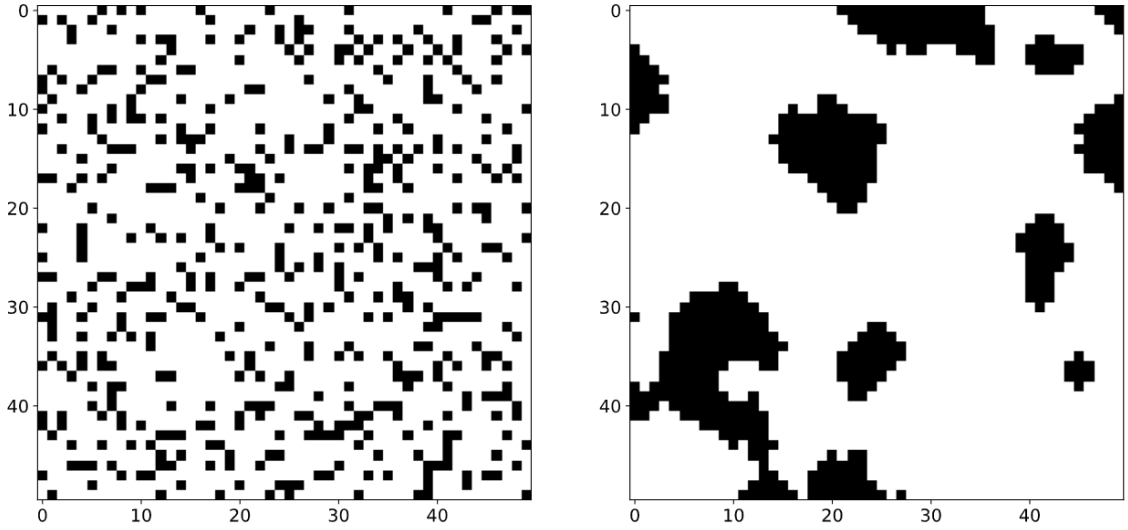
Rysunek 9: Wynik zastosowania symulowanego wyżarzana na losowym obrazie binarnym o gęstości $\delta = 0.2$

Stany sąsiednie możemy generować zamieniając losowe pixele ze sobą (rand_swap), albo zamieniać losowy pixel z którymś z jego sąsiadów (consecutive_swap), przy czym możemy rozpatrywać wiele rodzajów sąsiedztwa.

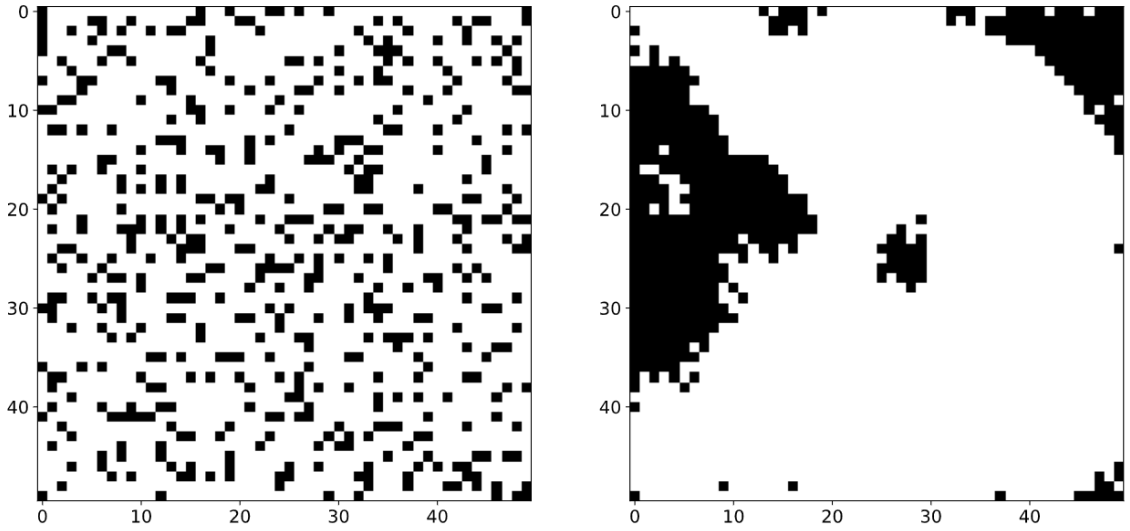
Wpływ rodzaju sąsiedztwa na uzyskane wyniki



Rysunek 10: Za sąsiadów bierzemy cztery bezpośrednio przystające pixele



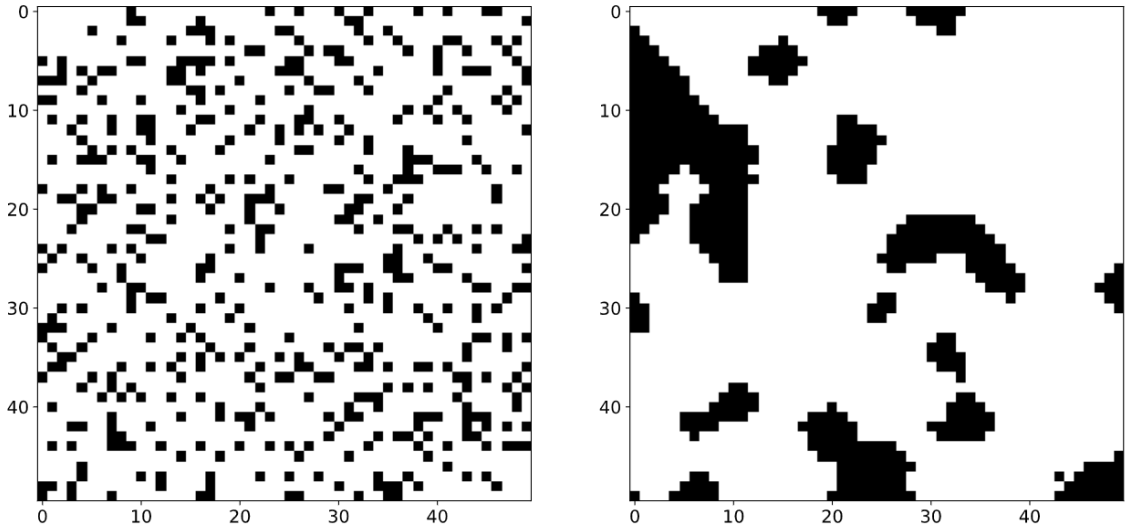
Rysunek 11: Za Sąsiadów bierzemy osiem przystających pixeli



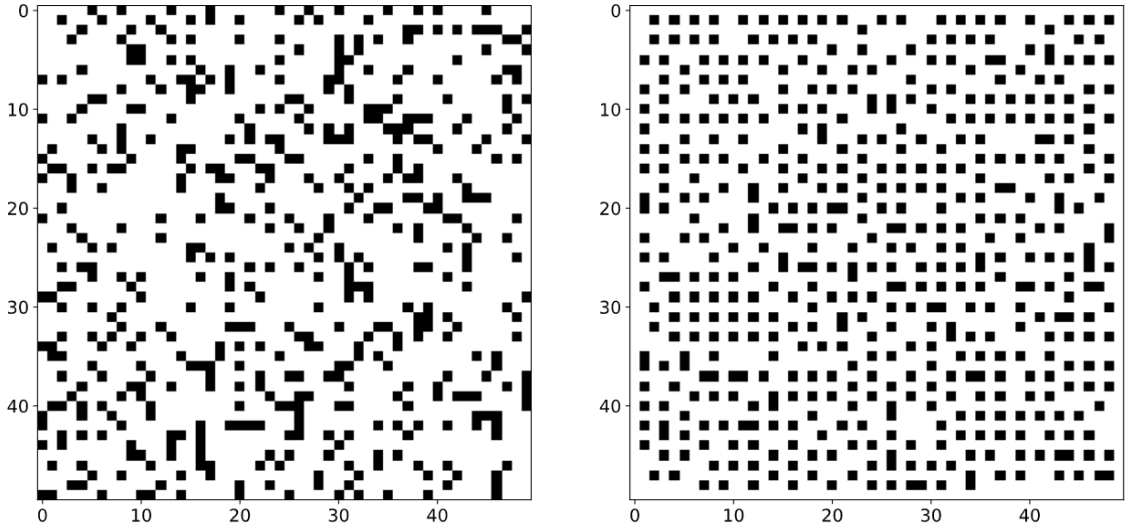
Rysunek 12: Tutaj za sąsiadów bierzemy też pixele przylegające do bezpośrednich sąsiadów (24)

Im więcej sąsiadów bierzemy pod uwagę tym większe powstają skupiska czarnych pól. Dzieje się tak dlatego, że wykorzystywana funkcja kosztu bierze pod uwagę większy obszar, przez co algorytm minimalizując ją będzie dążył do utworzenie możliwie jak największych skupisk - bo koszt będzie wtedy niższy niż wiele małych wysepek. Gdy sąsiadów jest mało wiele wysepek dobrze minimalizuje funkcję kosztu - stąd różnice w wynikach.

Wpływ wybranej funkcji energii



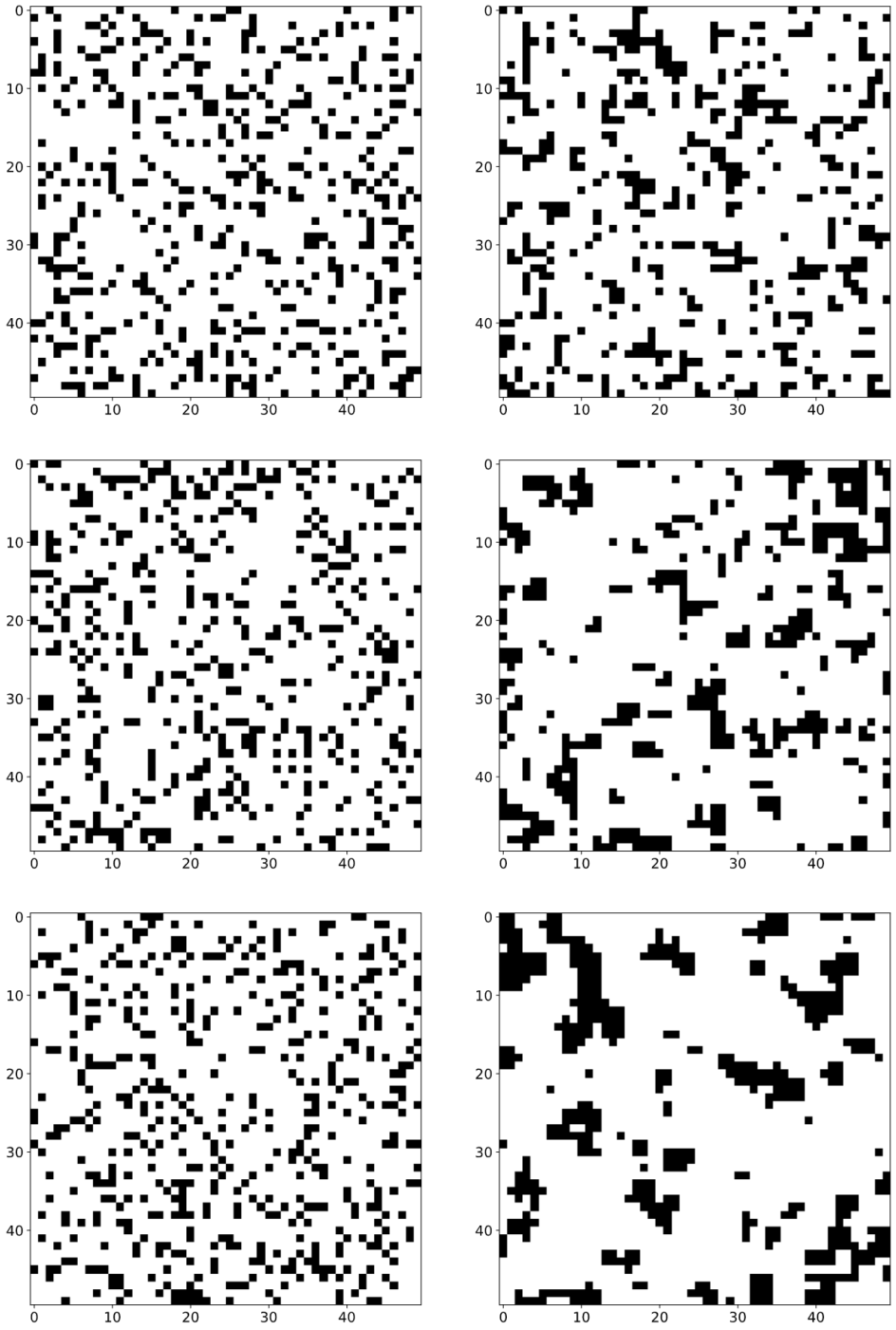
Rysunek 13: Pixele tego samego rodzaju się przyciągają



Rysunek 14: Pixele tego samego rodzaju się odpychają

Funkcja energii kształtuje nasze rozwiązanie - jakakolwiek ona nie będzie algorytm będzie starał się ją zminimalizować więc możemy dostać bardzo różnie wyglądające rozwiązania zmieniając tylko funkcję kosztu.

Wpływ funkcji temperatury na rozwiązanie



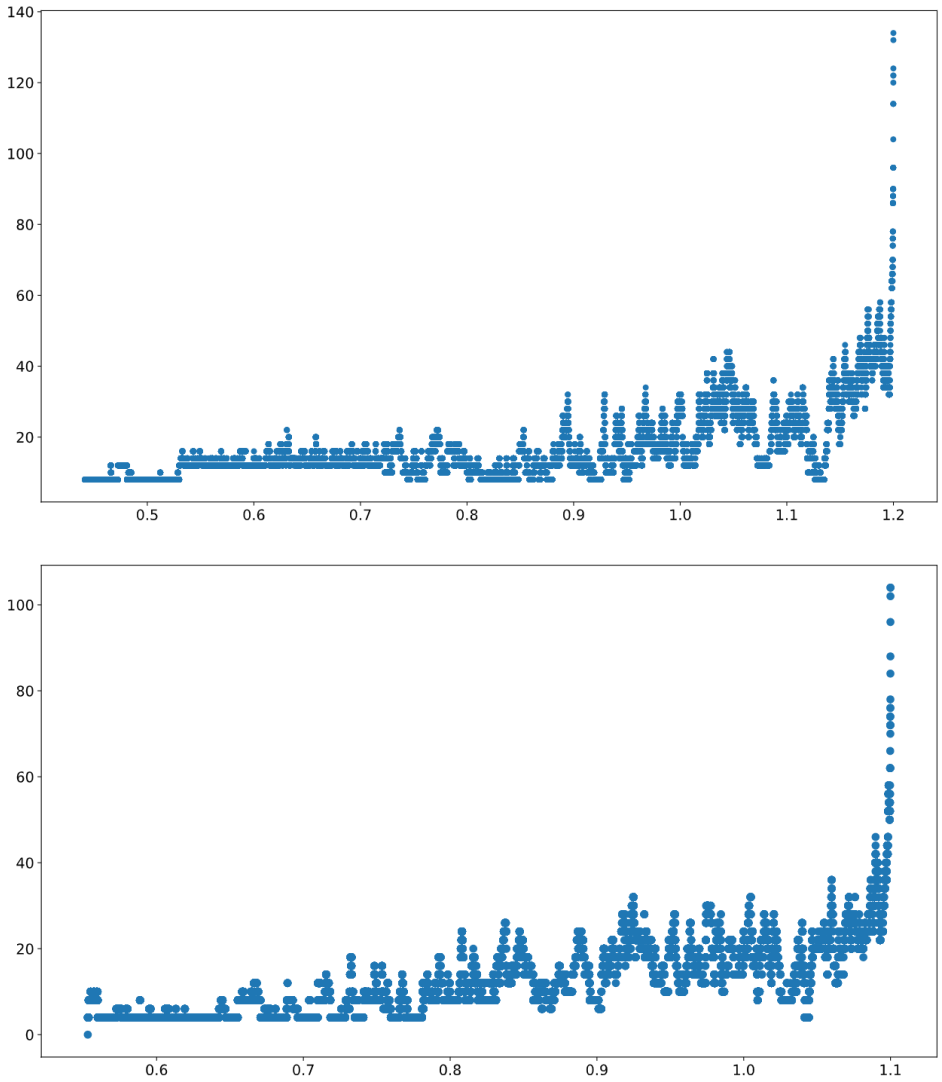
Powyżej wykorzystano tą samą funkcję kosztu, jak i dobór sąsiedztwa - obrazy różnią się jedynie wykorzystaną funkcją temperatury. Pierwsza spada bardzo drastycznie, ostatnia bardzo wolno. Widać, że im szybkość spadania temperatury jest niższa tym algorytm ma większą szansę na znalezienie lepszego rozwiązania.

Zadanie 3 - solver sudoku

W tym zadaniu szczególnie trudno było dobrać odpowiednie paramtetry symulacji. Ostatecznie za funkcję tempertury obralem taką która ochładza system bardzo wolno:

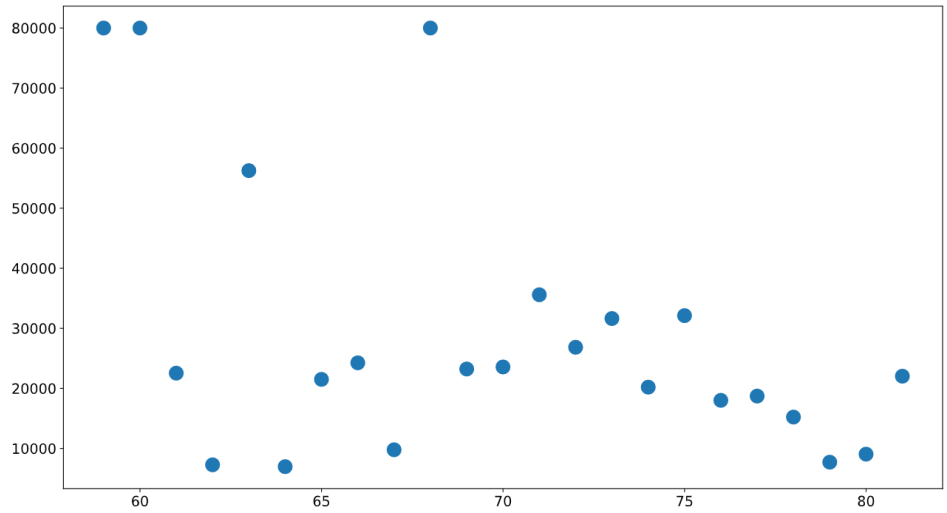
$$tempF(x) = 0.99999 \cdot x$$

Funkcja generująca stan następny wybiera losową kolumnę i zamienia miejscami dwie cyfry (oczywiście takie które nie były podane na wejściu). Z racji tego, że temperatura maleje bardzo wolno, nie robimy wielu iteracji dla tej samej temperatury - 1 epoka dla 1 temperatury. Dla tego problemu algorytm bardzo łatwo wpada w lokalne minimum więc nie zawsze udaje mu się znaleźć rozwiązanie (przynajmniej w ograniczonym czasie).



Wykresy przedstawiają zależność kosztu od temperatury. Na wykresach widać, że algorytm trafia na minimum i przez długi czas nie może z niego wyjść, czasem w ogóle mu się to nie udaje.

Podstawową trudnością jest specyfika problemu. W przypadku tsp rozwiązanie w okolicach 2-3% od optymalnego rozwiązania było dla nas satysfakcjonujące. Tutaj natomiast rozwiązaniem jest globalne minimum co algorytmom aproksymacyjnym rzadko się udaje.



Wykres przedstawia zależność ilości potrzebnych iteracji do znalezienia rozwiązania od brakujących pól. Wykres wykonałem tylko dla niewielkich wartości brakujących pól, gdyż czas działania był zbyt długi (ucinam szukanie rozwiązania gdy proces przekroczy 80000 iteracji). Wraz z rosnącą ilością brakujących pól rośnie (dość drastycznie) ilość iteracji potrzebnych do znalezienia rozwiązania. Trzeba też zwrócić uwagę, że algorytm nie jest "konsekwentny" tzn. czasem uda mu się znaleźć rozwiązanie bardzo szybko, a czasem utknie w jakimś miejscu (lokalnym minimum - być może bardzo odległym od faktycznego rozwiązania).