

Mownit 2

Sprawozdanie

Zadanie 4 – symulowane wyżarzanie

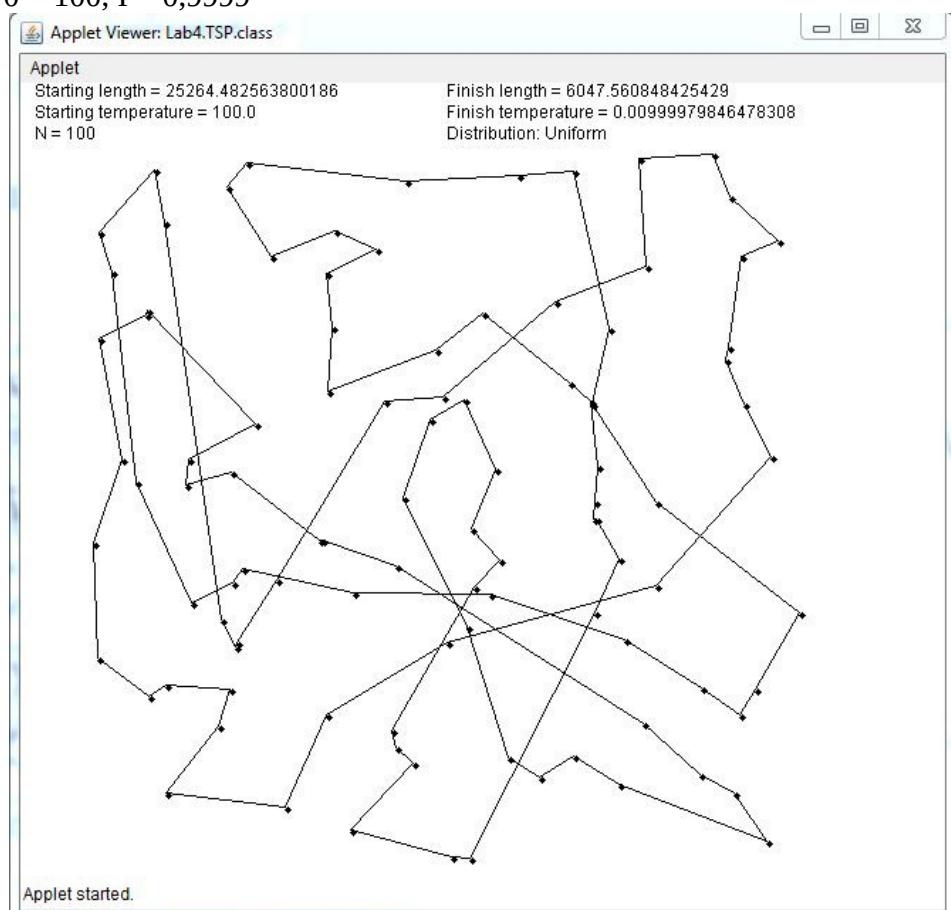
1. Rozwiążanie problemu komiwojażera dla wygenerowanych chmur N punktów.

a) Poniżej przedstawione są wizualizacje rozwiązań dla różnych N oraz różnych dystrybucji punktów.

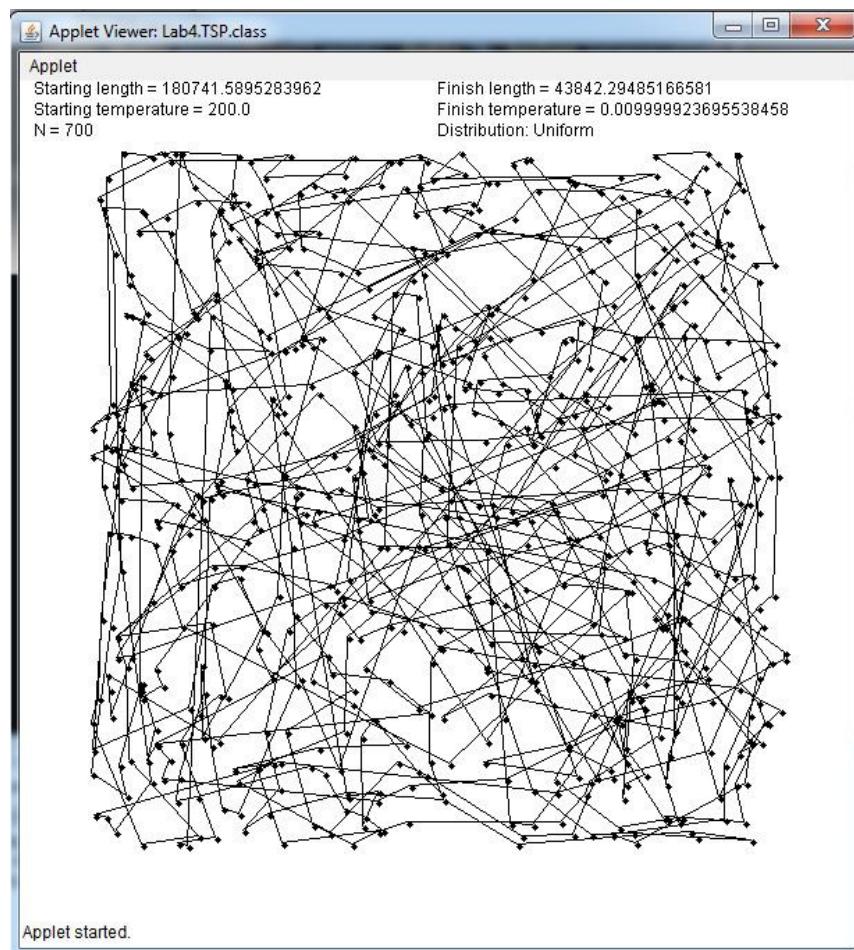
Oznaczenia: N – ilość wygenerowanych punktów, T₀ – początkowa temperatura, I – współczynnik wyżarzania we wzorze $T(n+1) = I * T(n)$ (wykładnicza funkcja zmiany temperatury), μ – wartość średnia (użyta przy tworzeniu rozkładu normalnego), $\sigma = 300$ – odchylenie standardowe (użyte przy tworzeniu rozkładu normalnego)

Rozkład jednostajny

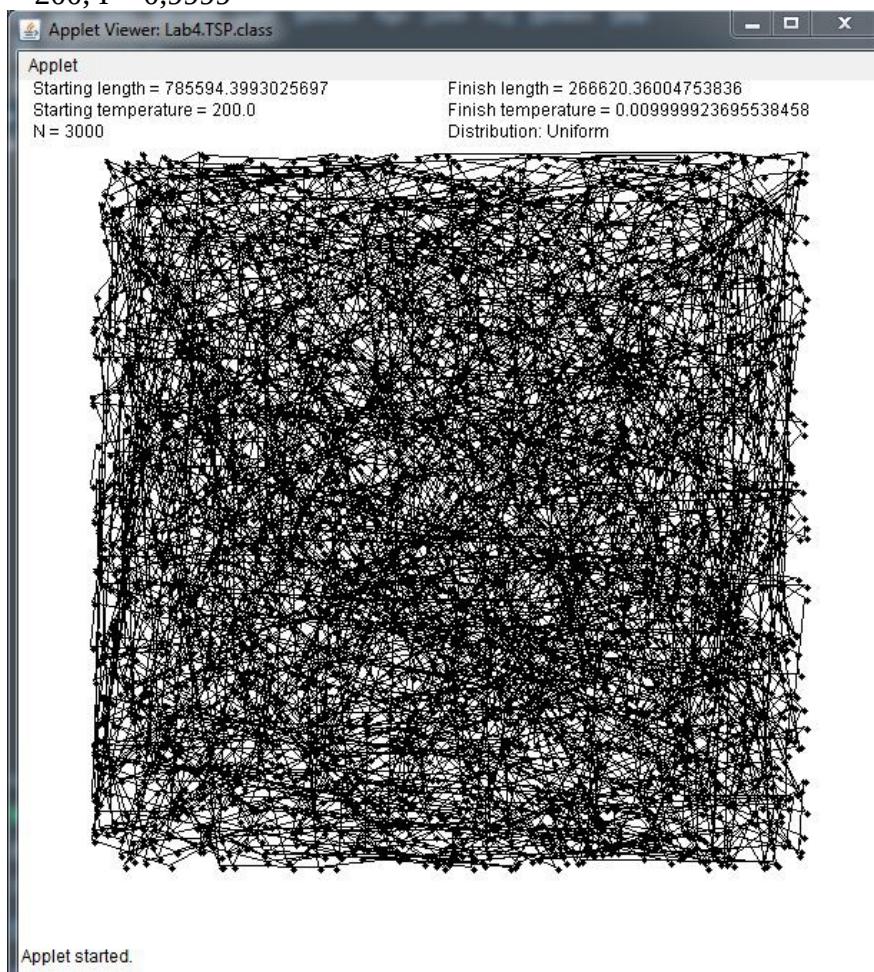
* N = 100, T₀ = 100, I = 0,9999



* N = 700, T0 = 200, I = 0,9999

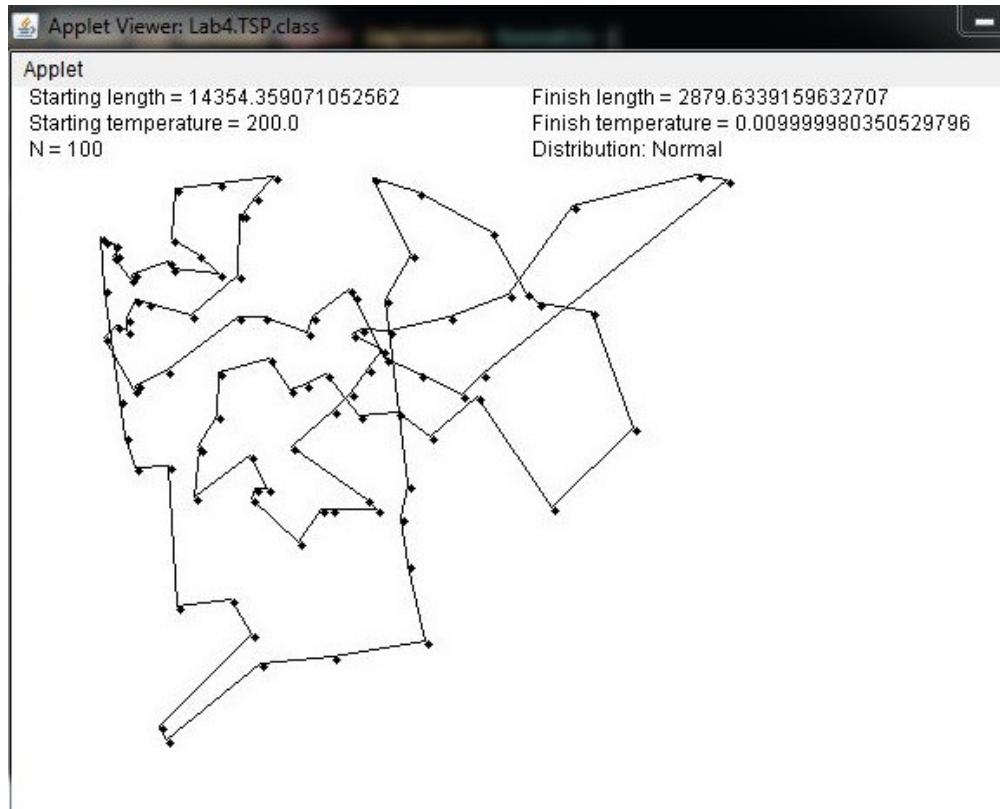


* N = 3000, T0 = 200, I = 0,9999

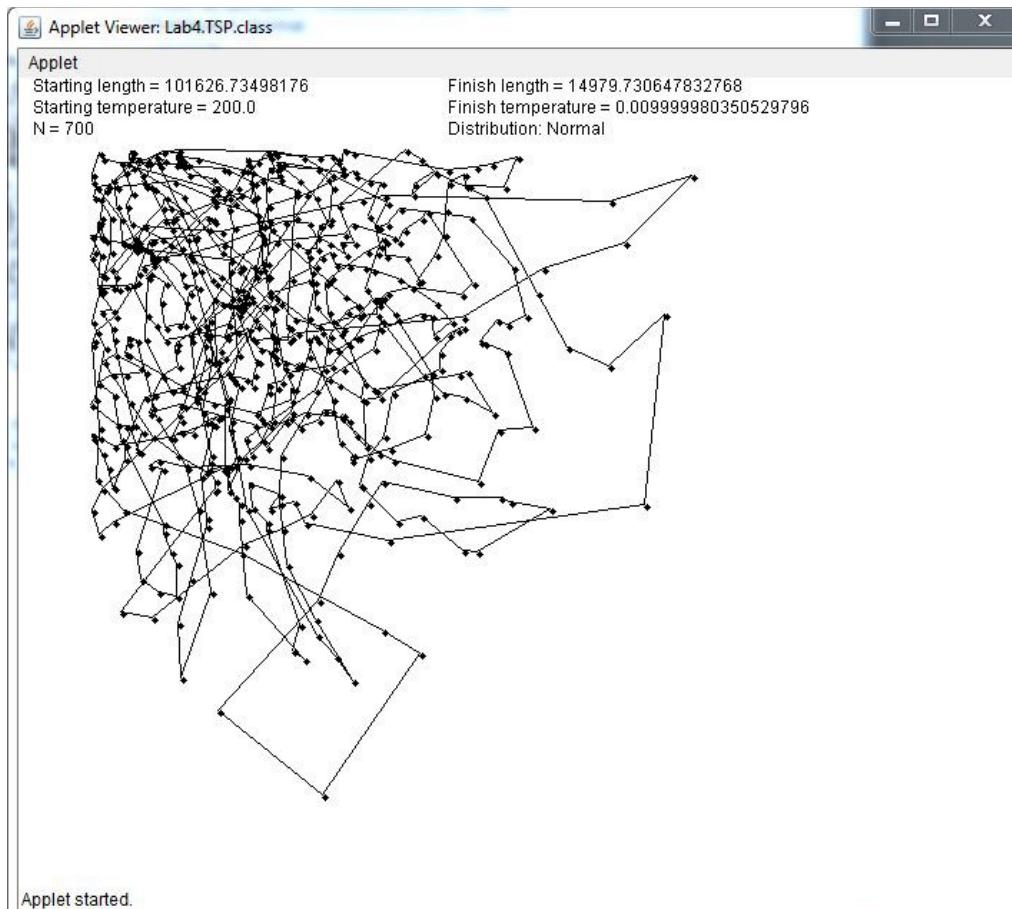


Rozkład normalny

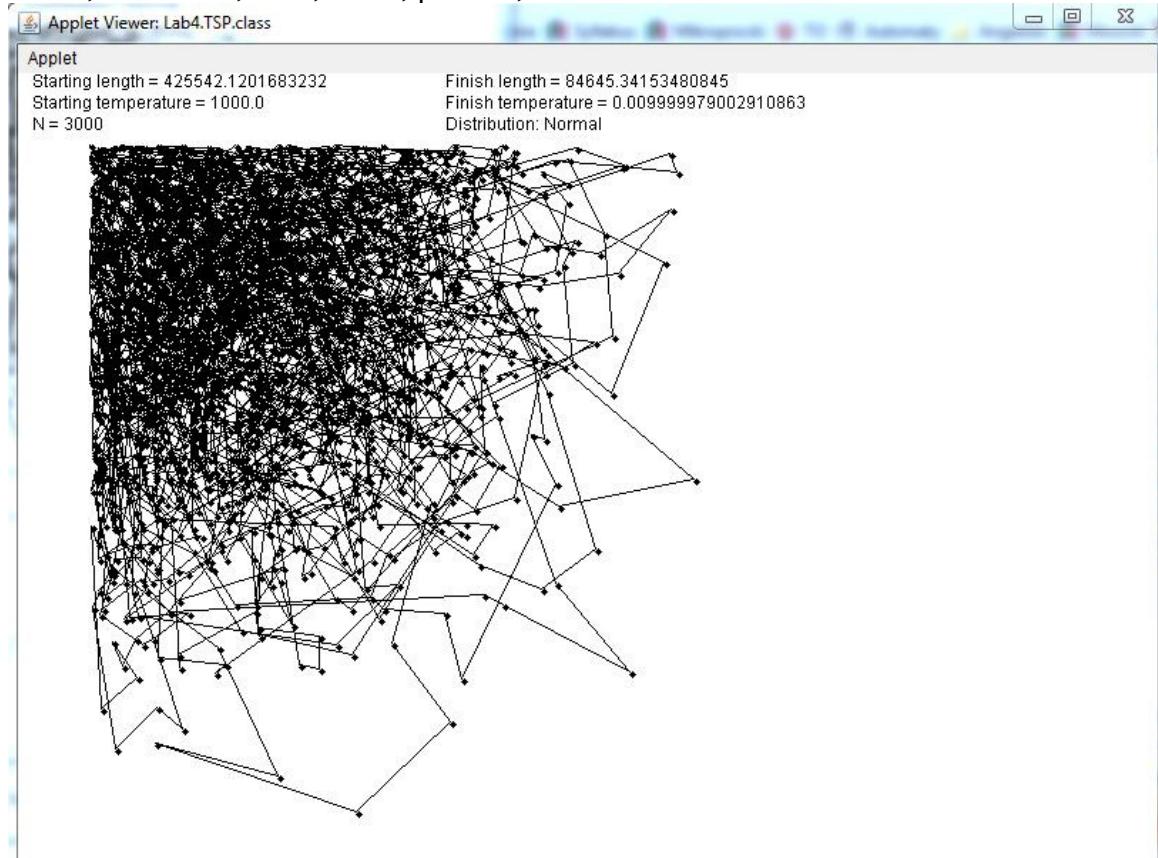
* $N = 100$, $T_0 = 200$, $I = 0,99999$, $\mu = 100$, $\sigma = 100$.



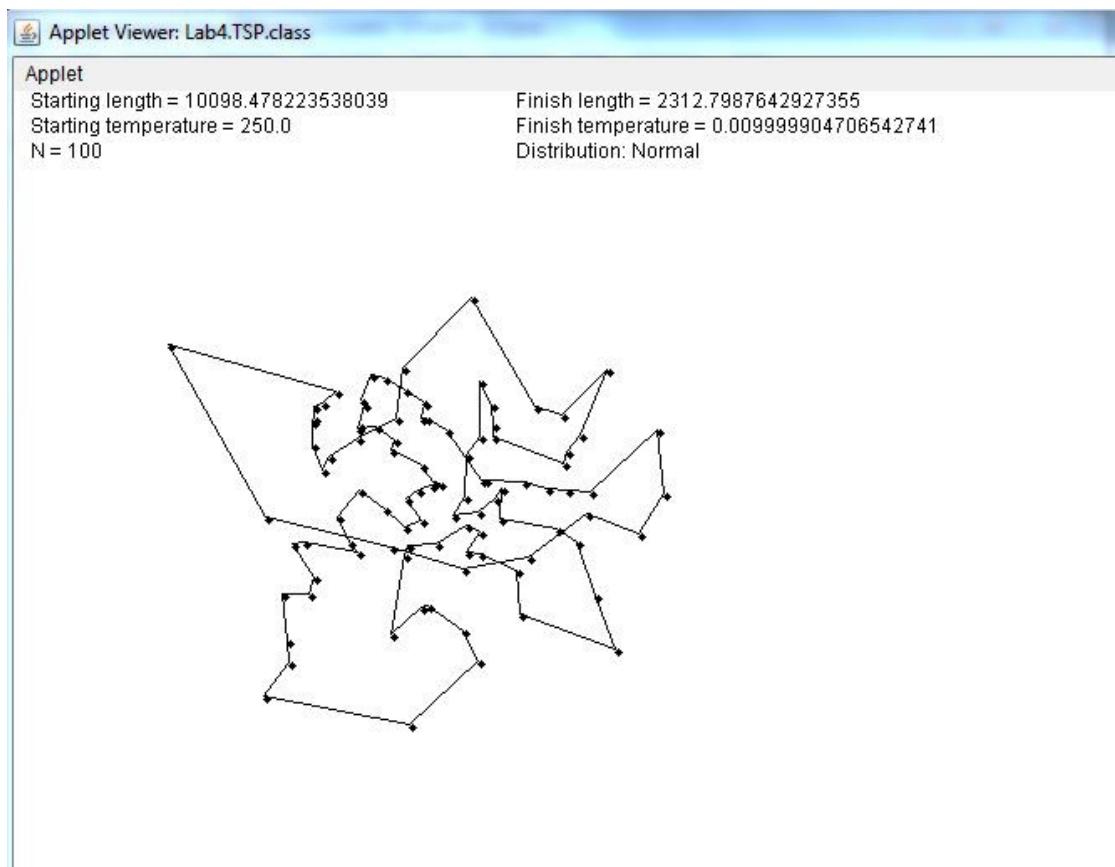
* $N = 700$, $T_0 = 200$, $I = 0,99999$, $\mu = 100$, $\sigma = 100$.



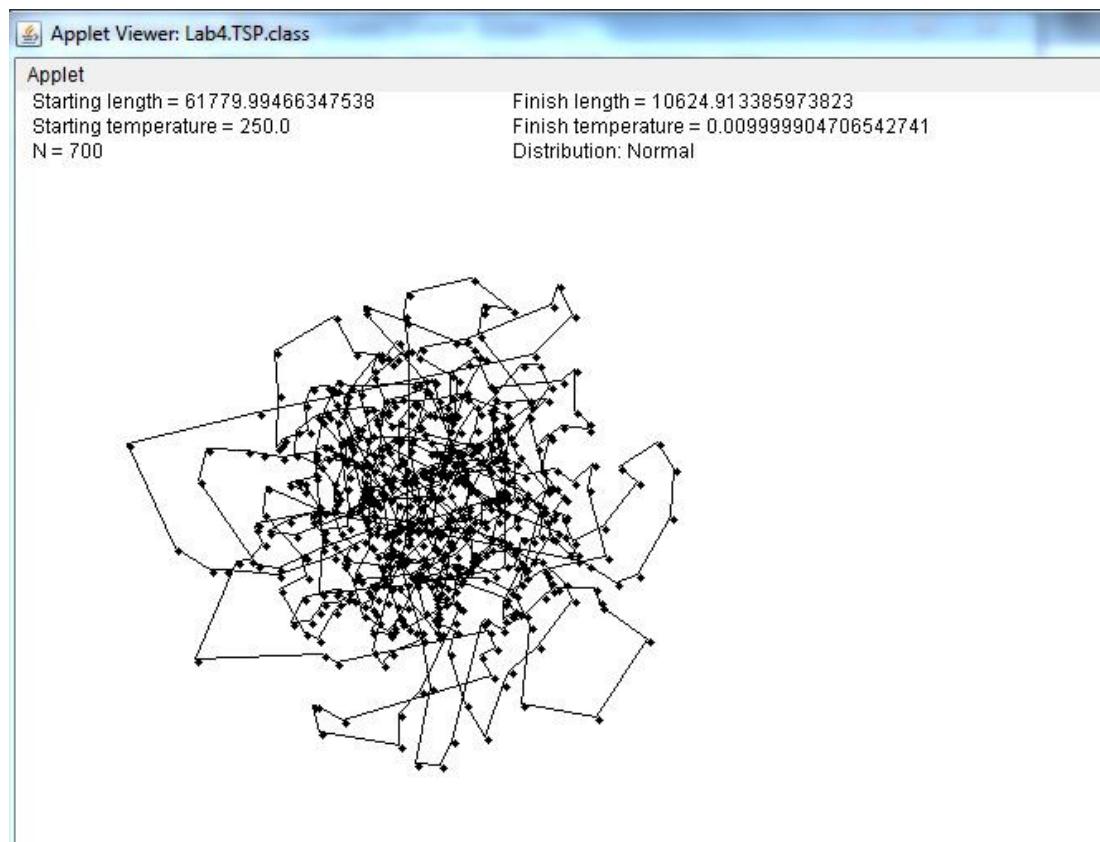
* $N = 3000$, $T_0 = 1000$, $I = 0,99999$, $\mu = 100$, $\sigma = 100$.



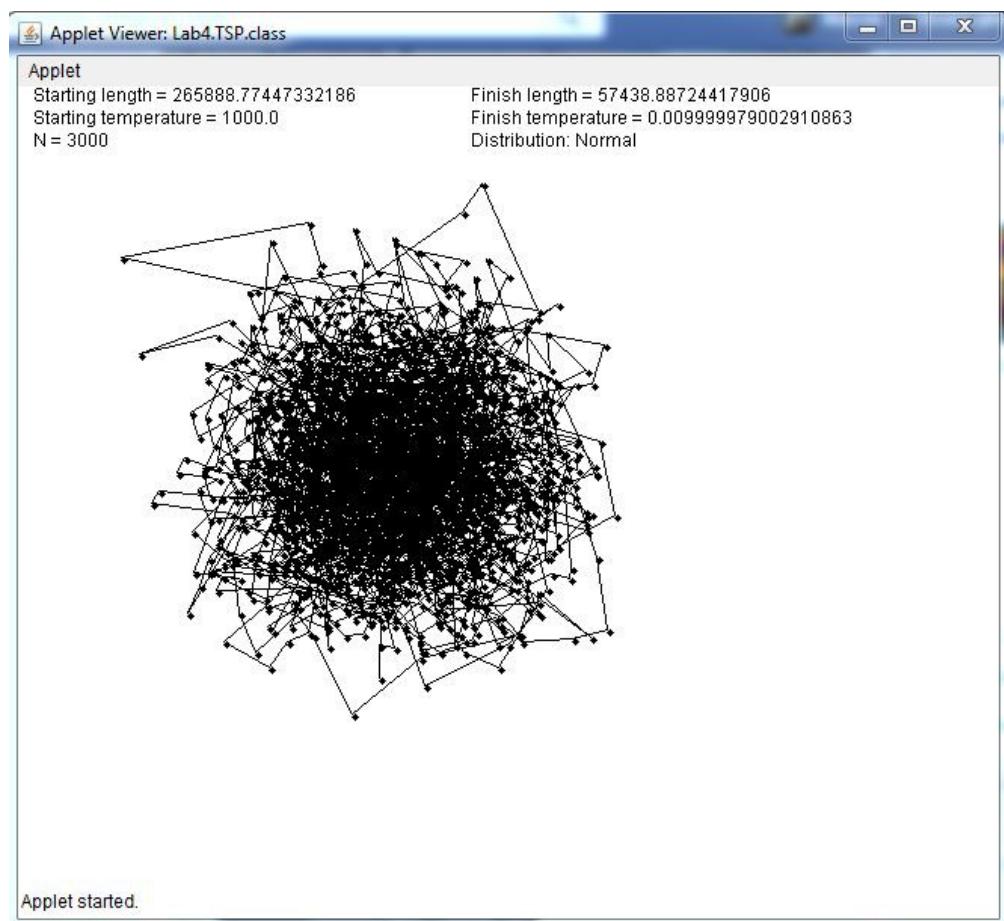
* $N = 100$, $T_0 = 250$, $I = 0,99999$, $\mu = 200$, $\sigma = 50$.



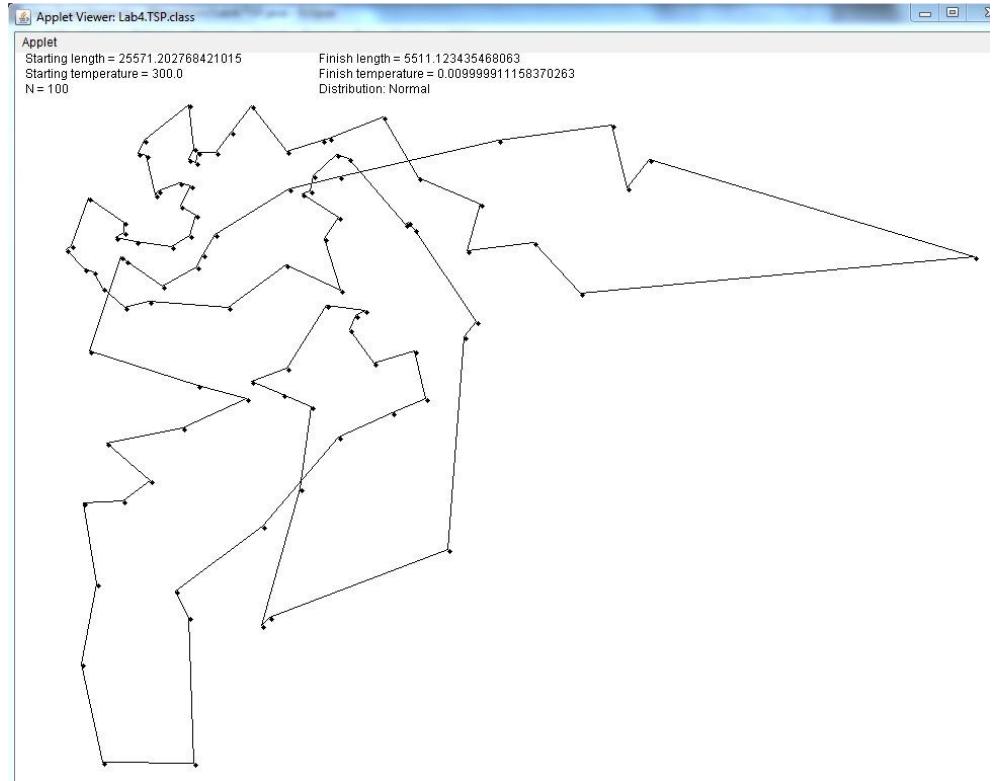
* N = 700, T0 = 250, I = 0,99999, μ = 200, σ = 50.



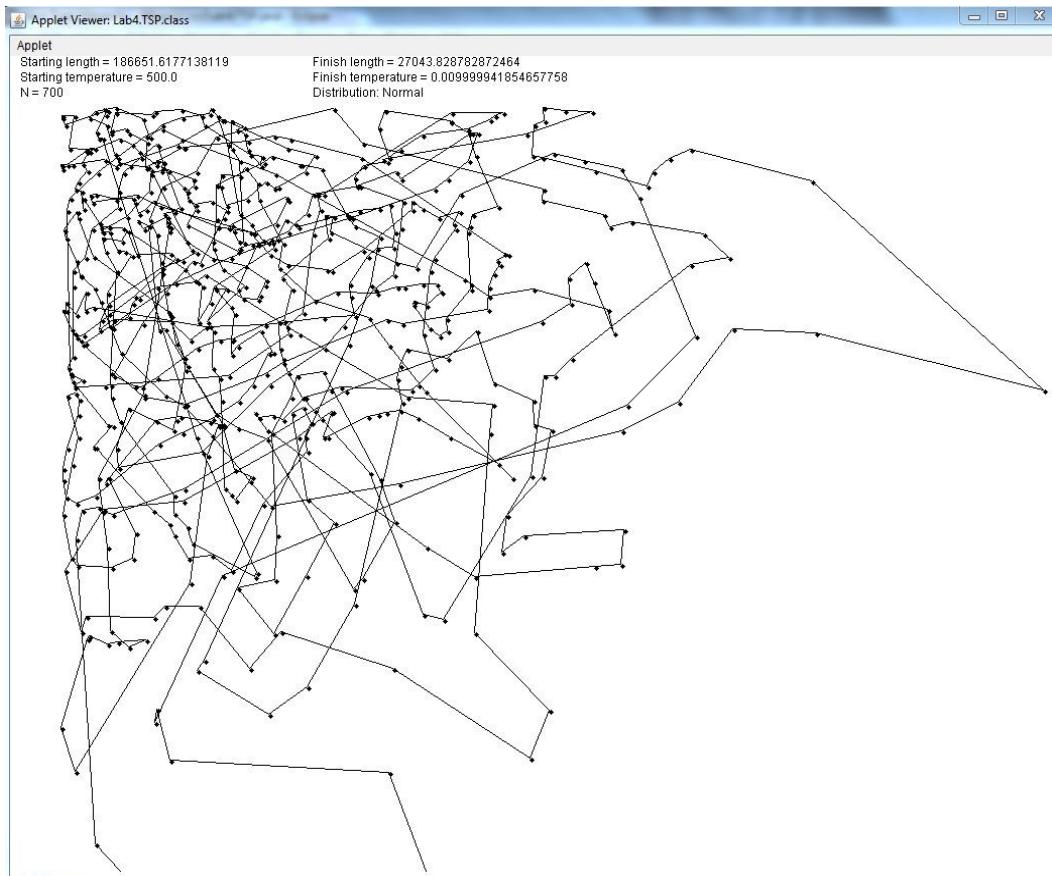
* N = 3000, T0 = 1000, I = 0,99999, μ = 200, σ = 50.



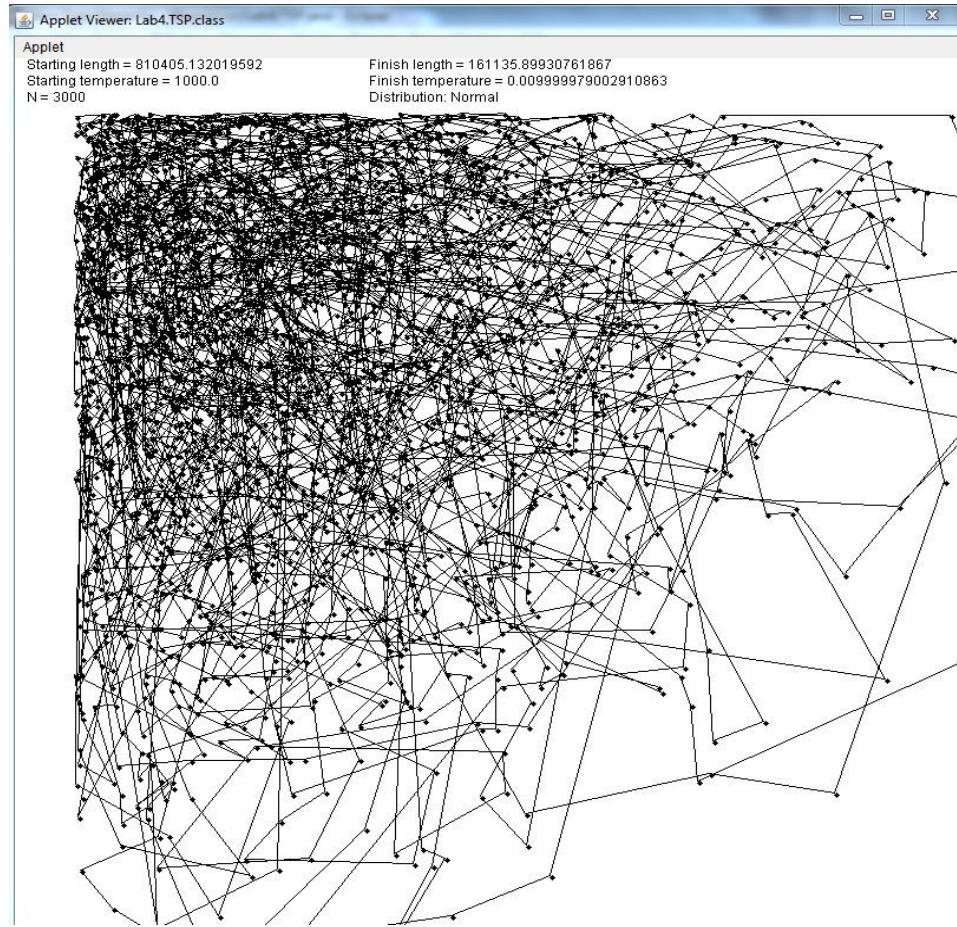
* $N = 100$, $T_0 = 300$, $I = 0,99999$, $\mu = 50$, $\sigma = 250$.



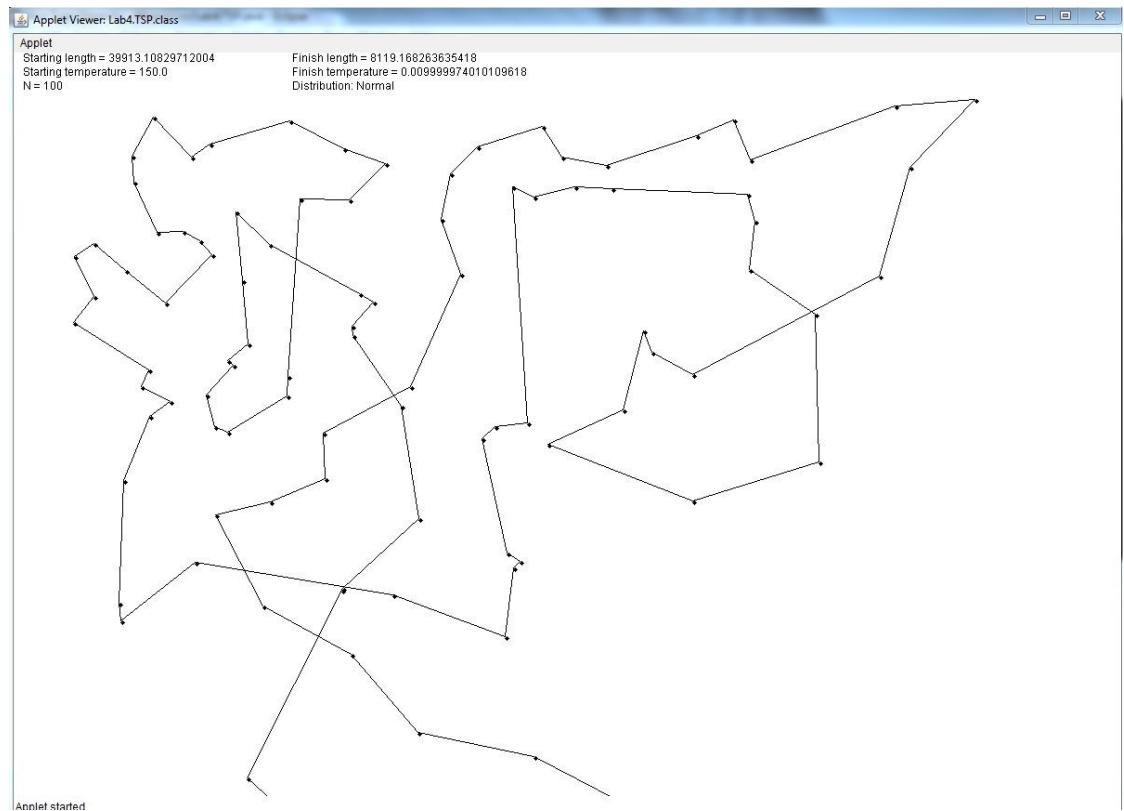
* $N = 700$, $T_0 = 500$, $I = 0,99999$, $\mu = 50$, $\sigma = 250$.



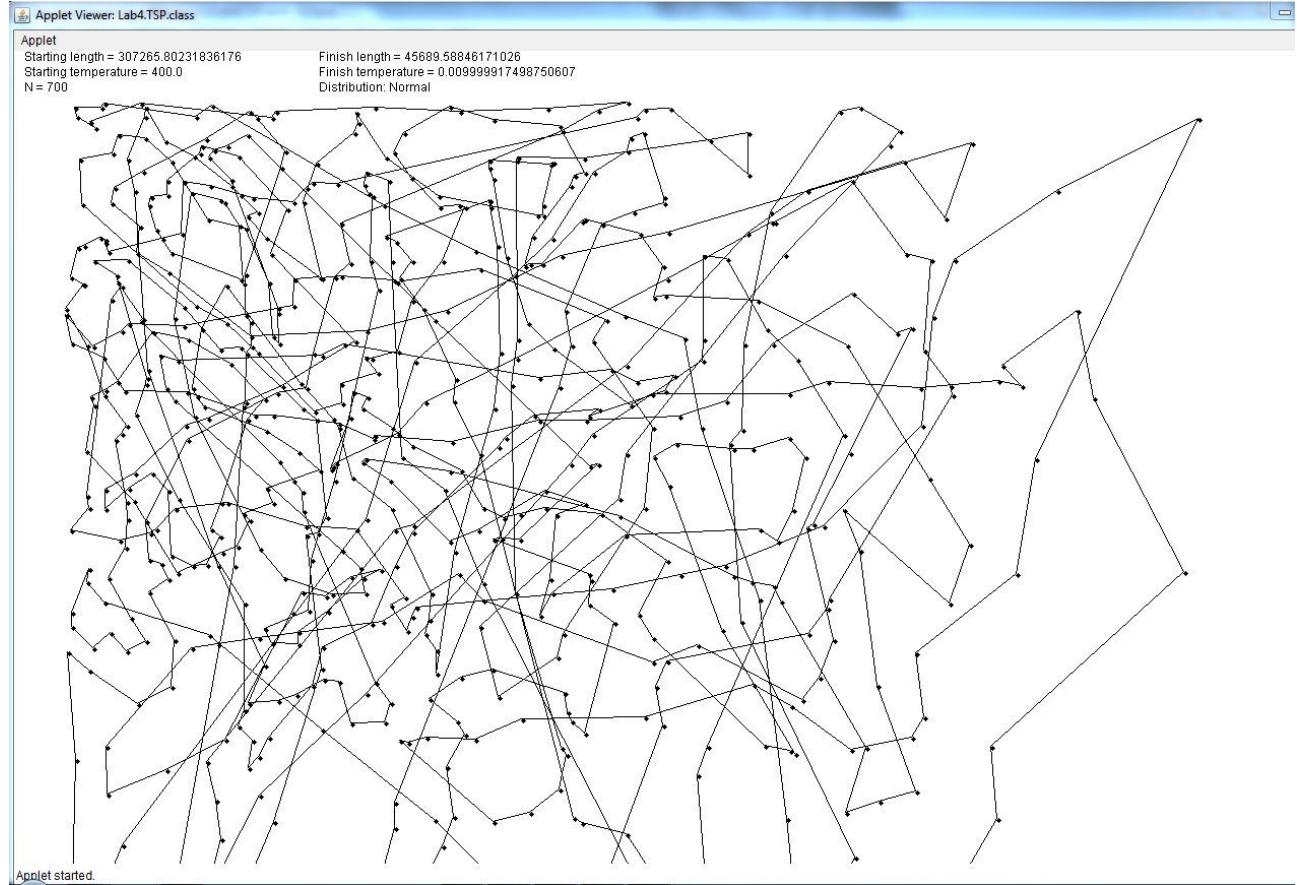
* $N = 3000$, $T_0 = 1000$, $I = 0,99999$, $\mu = 50$, $\sigma = 250$.



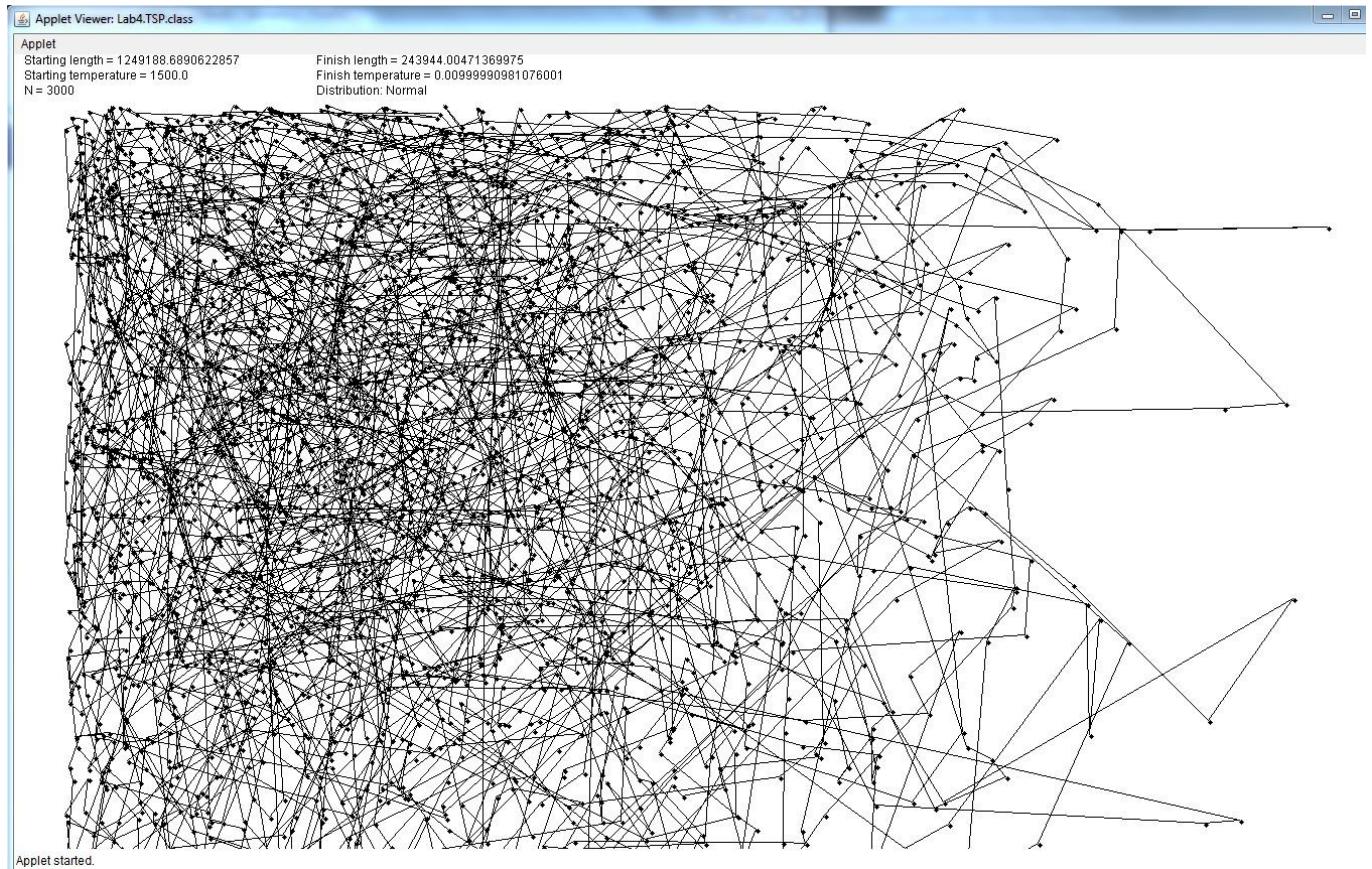
* $N = 100$, $T_0 = 150$, $I = 0,99999$, $\mu = 300$, $\sigma = 300$.



* N = 700, T0 = 400, I = 0,99999, μ = 300, σ = 300.

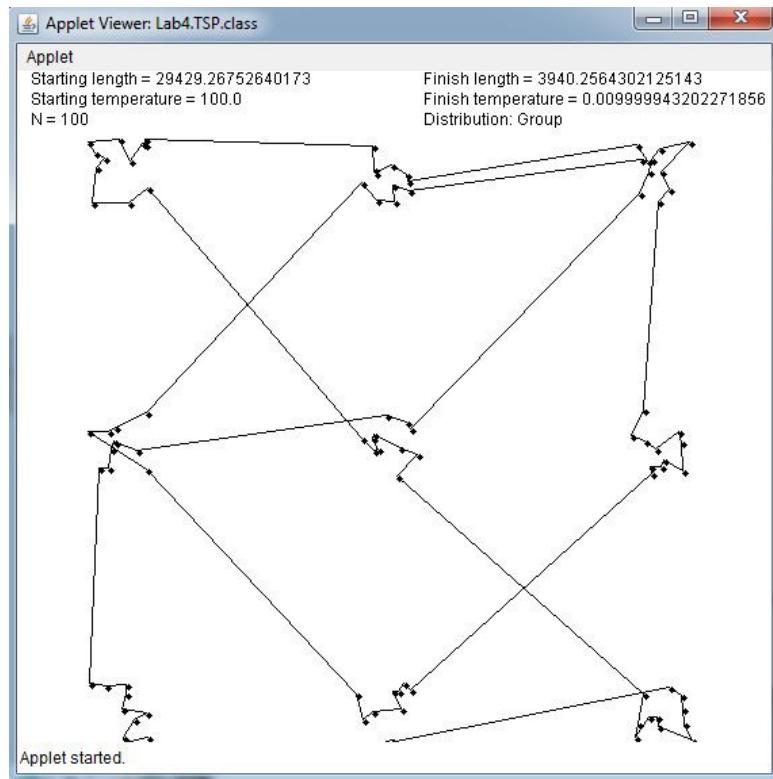


* N = 3000, T0 = 1500, I = 0,99999, μ = 300, σ = 300.

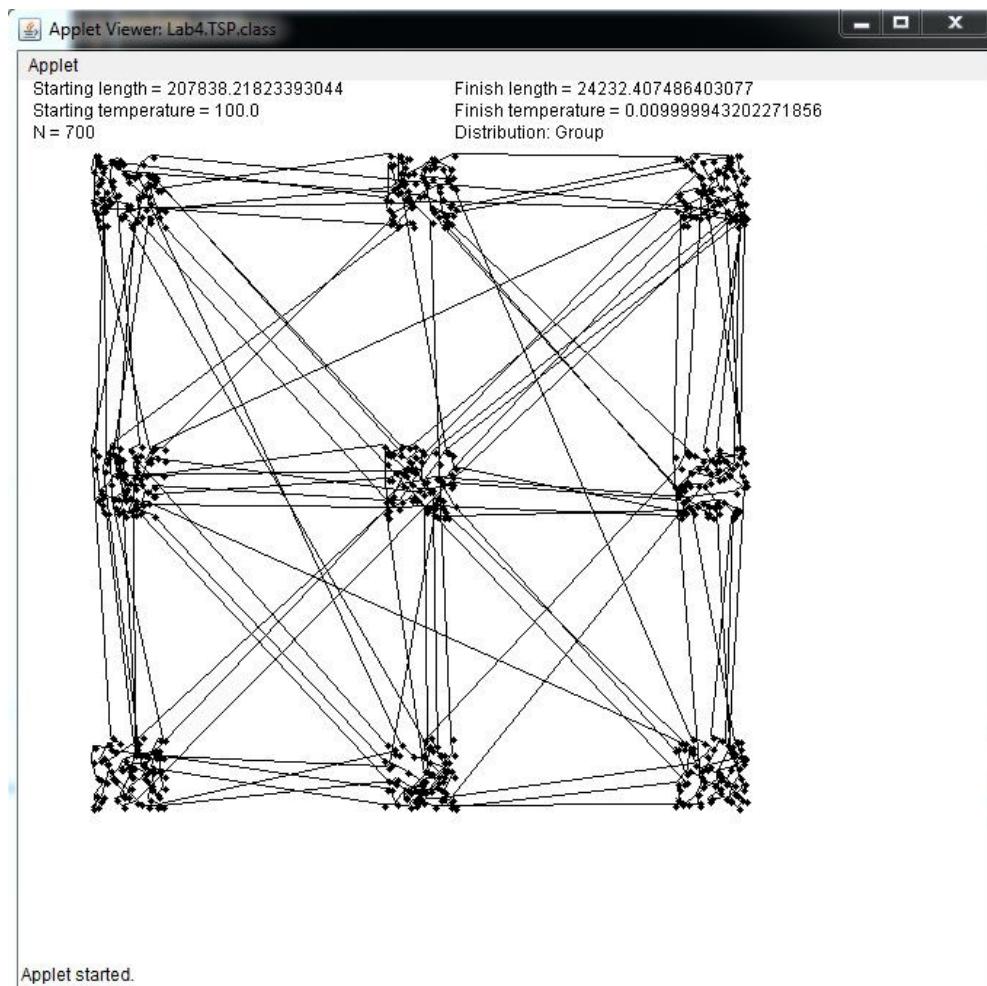


Rozkład grupowy:

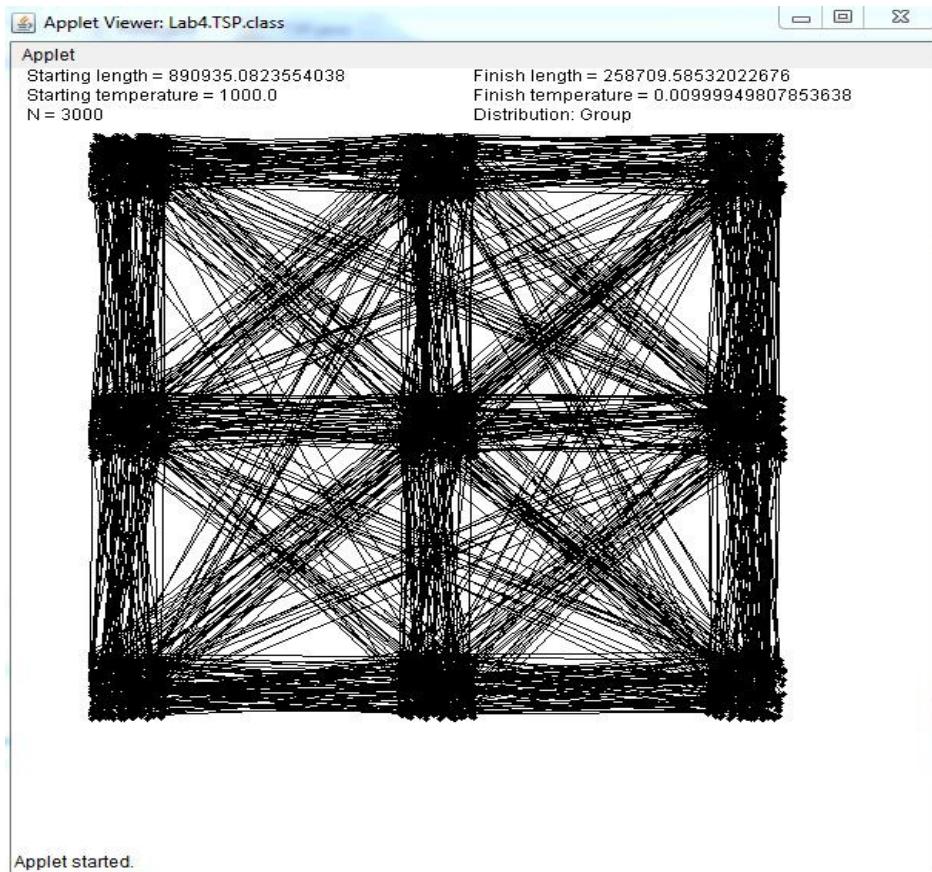
* N = 100, T₀ = 100, I = 0,99999



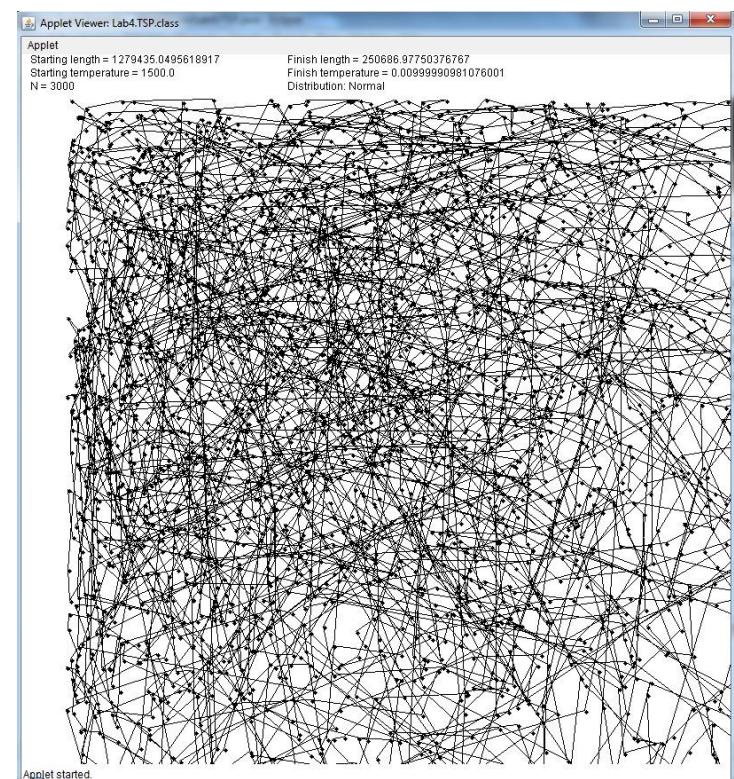
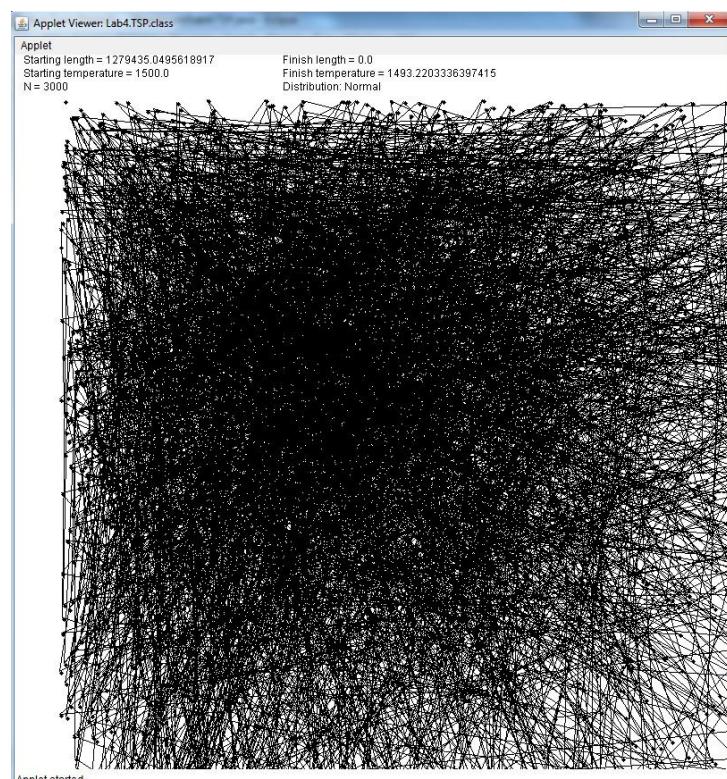
* N = 700, T₀ = 100, I = 0,99999



* N = 3000, T0 = 2000, I = 0,9999

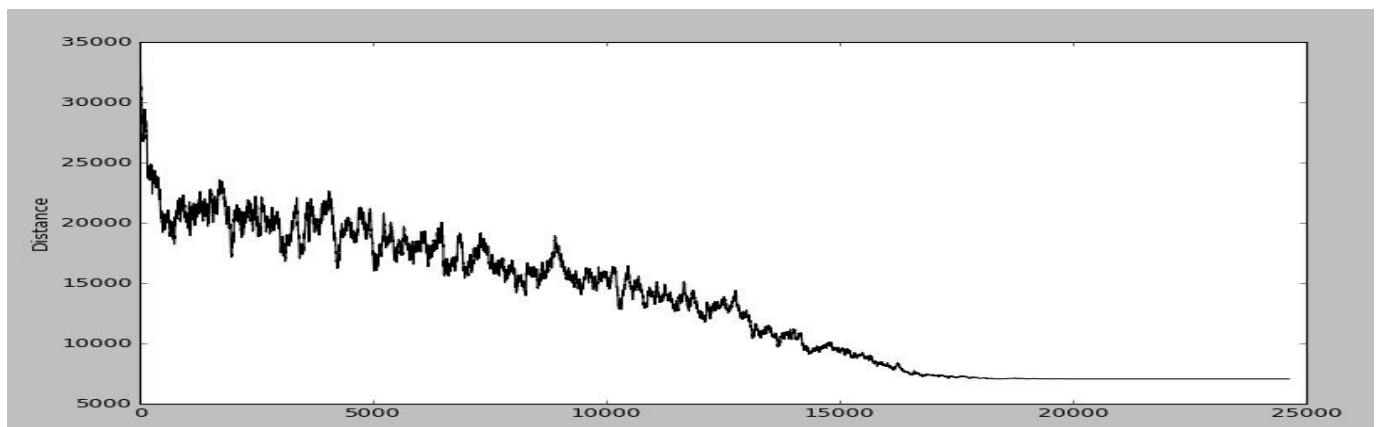


Wizualizacje podpunktów, gdzie N = 3000 mogą niewystarczająco prezentować zoptymalizowanie ścieżki, ze względu na dużą gęstość punktów. Dlatego do każdej wizualizacji została dodana informacja o długości ścieżki przed zastosowaniem algorytmu oraz po zastosowaniu. Różnice są znaczące, oscylują w okolicy 5 razy mniejszej wartości po wykonaniu algorytmu. Oto porównanie wyglądu ścieżki losowo wygenerowanej (przed rozpoczęciem działania algorytmu) oraz po zastosowaniu algorytmu dla N = 3000 i dystrybucji normalnej:

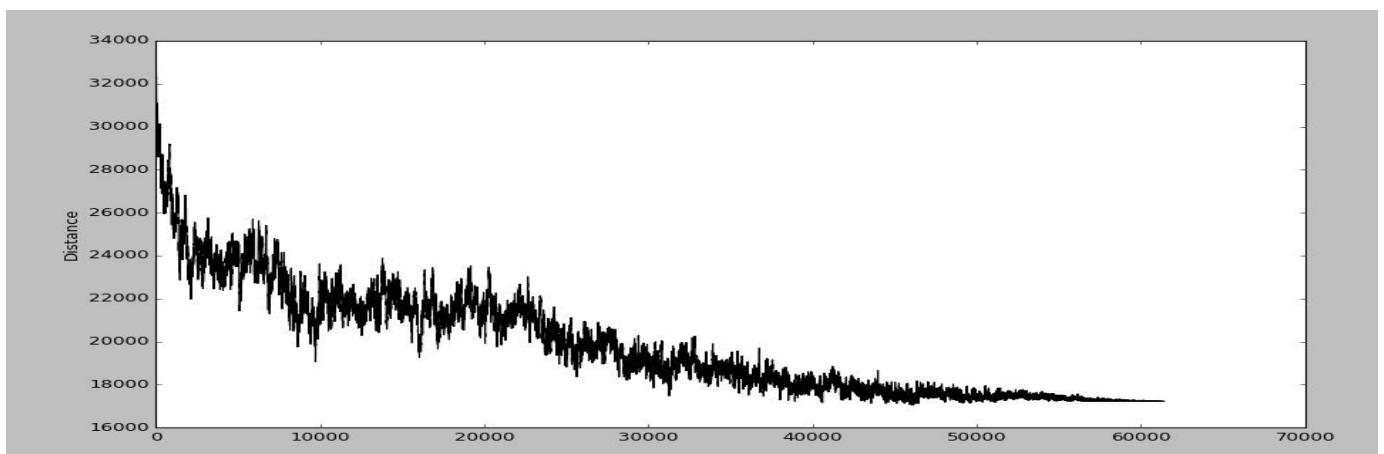


b) Poniżej przedstawione są wizualizacje oraz wnioski z badań wpływu sposobu generacji sąsiedniego stanu na zbieżność procesu optymalizacji. Rozpatrzony został arbitrary swap oraz consecutive swap. Pierwszy z nich polega na zamianie dwóch losowych punktów w tablicy, która stanowi kolejność punktów w ścieżce. Consecutive swap polega na zamianie sąsiednich punktów w tejże tablicy.

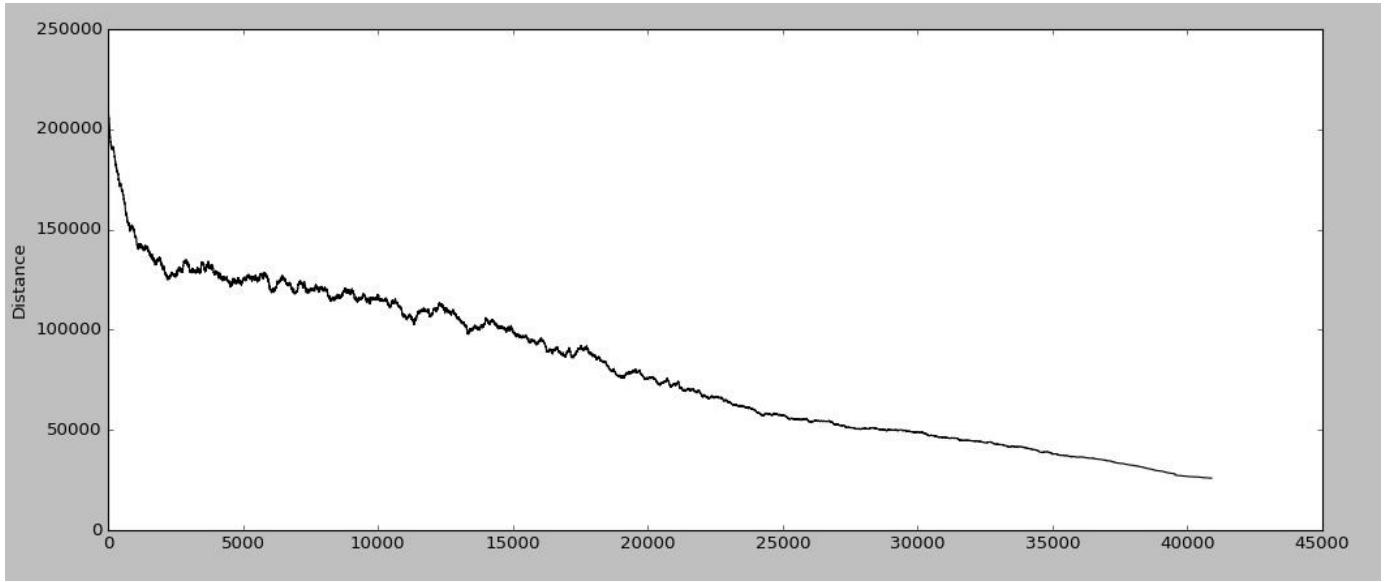
Rozkład jednostajny, $N = 100$, arbitrary swap:



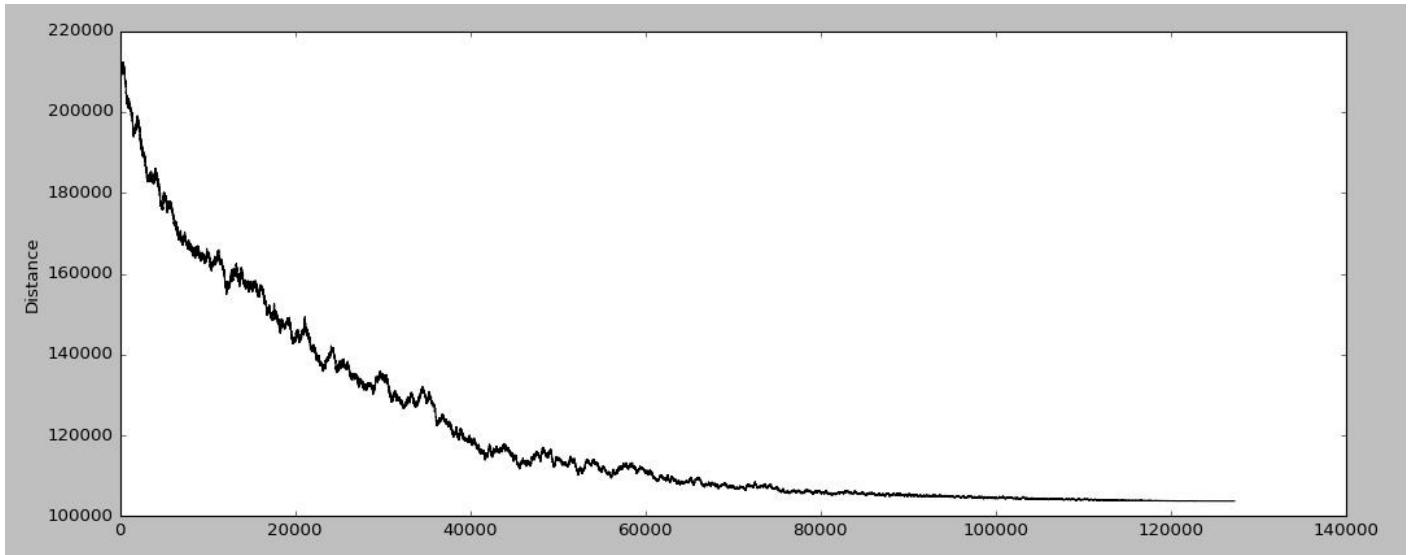
Rozkład jednostajny $N = 100$, consecutive swap:



Rozkład grupowy, N = 700, arbitrary swap:



Rozkład grupowy, N = 700 consecutive swap:



Analiza powyższych przykładów prowadzi do wniosku, że consecutive swap charakteryzuje się znacznie większymi wahaniemami wartości distance (funkcja, która jest minimalizowana) niż arbitrary swap. Tempo spadku wartości jest też bardziej zbliżone do wzorcowego, to znaczy: jest ono mniej zmienne. Początkowo spadek jest znaczny, ale z kolejnymi iteracjami jest coraz mniej wyraźny, zmiana tempa jest łagodna. W przypadku arbitrary swap następuje bardzo szybki spadek wartości distance w początkowych iteracjach a następnie tempo spadku utrzymuje się na zbliżonym poziomie. Mimo to w wizualizacjach prezentowanych poprzednio w podpunkcie 1a) oraz zaprezentowanych poniżej w 1c) zastosowany został arbitrary swap. Wiąże się to z faktem, iż końcowa wartość distance jest znacznie niższa, czyli mimo odstającego od wzorcowego przebiegu optymalizacji końcowy jej rezultat jest zdecydowanie lepszy niż przy stosowaniu consecutive swap.

Poniższe wizualizacje prezentują różnice w procesie optymalizacji przy zastosowaniu różnych rodzajów funkcji zmiany temperatury. Zastosowano dwie najczęściej spotykane rodzaje, czyli zmiana wykładnicza oraz liniowa:

I. wykładnicza, korzysta ze wzoru:

$$\text{currTemp} = \text{previousTemp} * \text{coolingRate}$$

gdzie:

currTemp to obecnie obliczana temperatura

previousTemp to poprzednia temperatura

coolingRate to współczynnik chłodzenia

startTemp to początkowa temperatura, z którą algorytm rozpoczął działanie

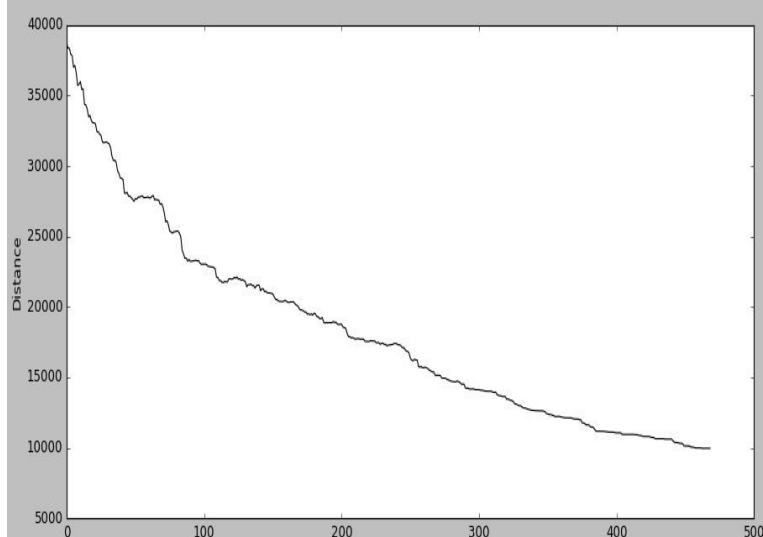
iteration to numer obecnej iteracji

II. Liniowa, korzysta ze wzoru:

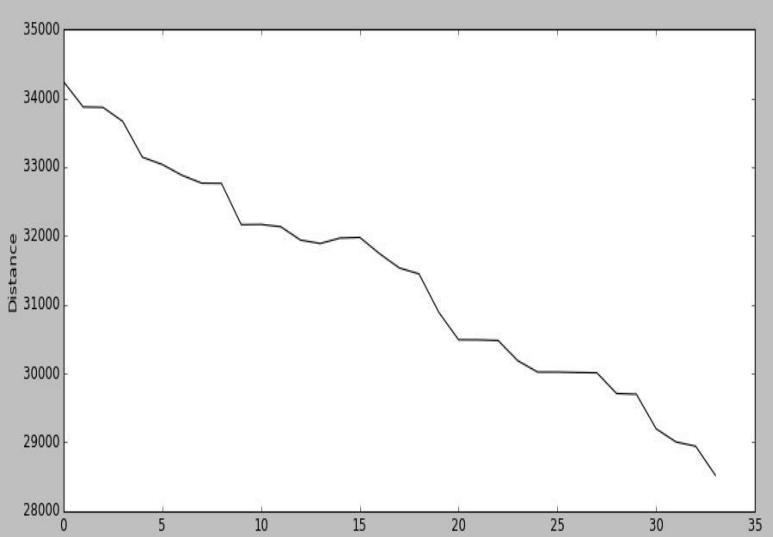
$$\text{currTemp} = \text{startTemp} - \text{coolingRate} * \text{iteration}$$

Rozkład jednostajny, $N = 100$, $\text{coolingRate} = 0.999$

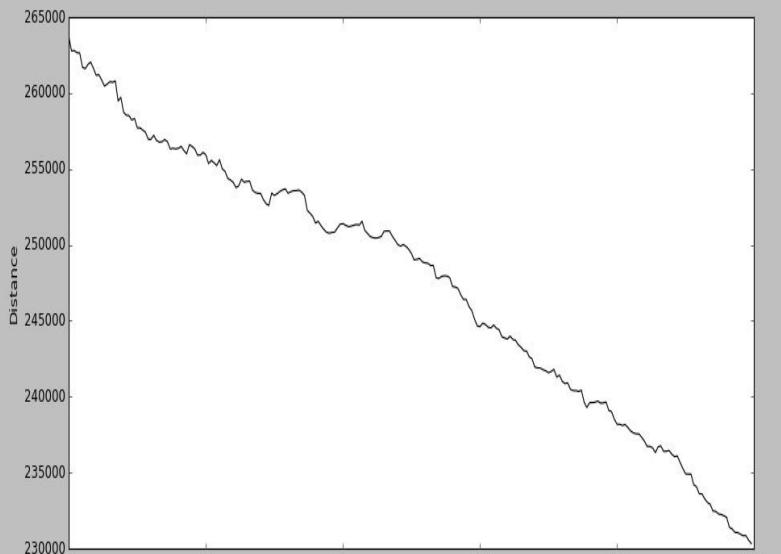
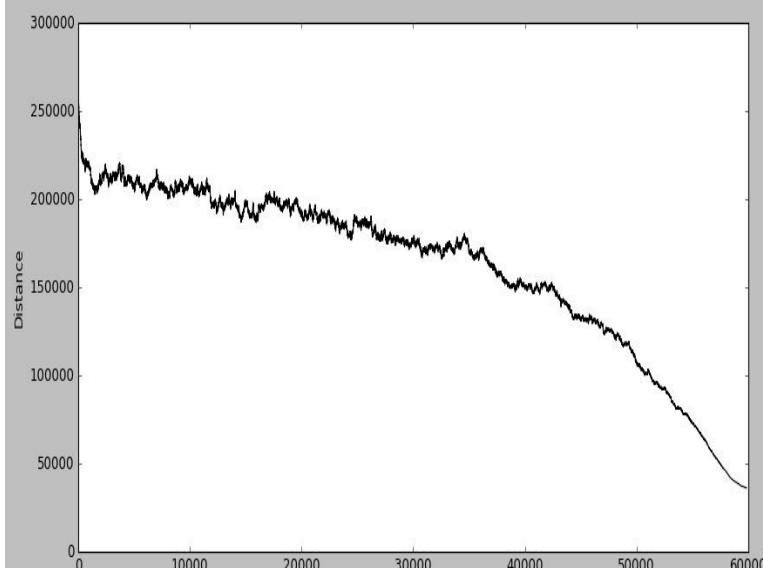
wykładnicza:



liniowa:



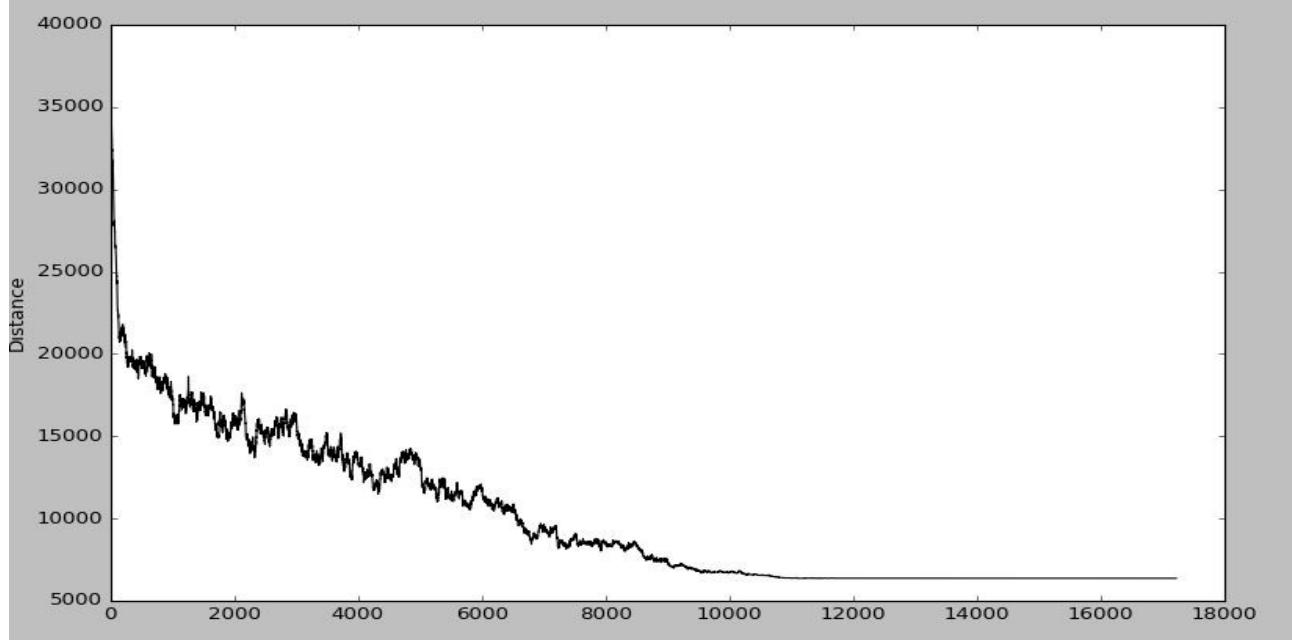
Rozkład normalny, $N = 700$, $\mu = 300$, $\sigma = 300$, $\text{coolingRate} = 0.99999$



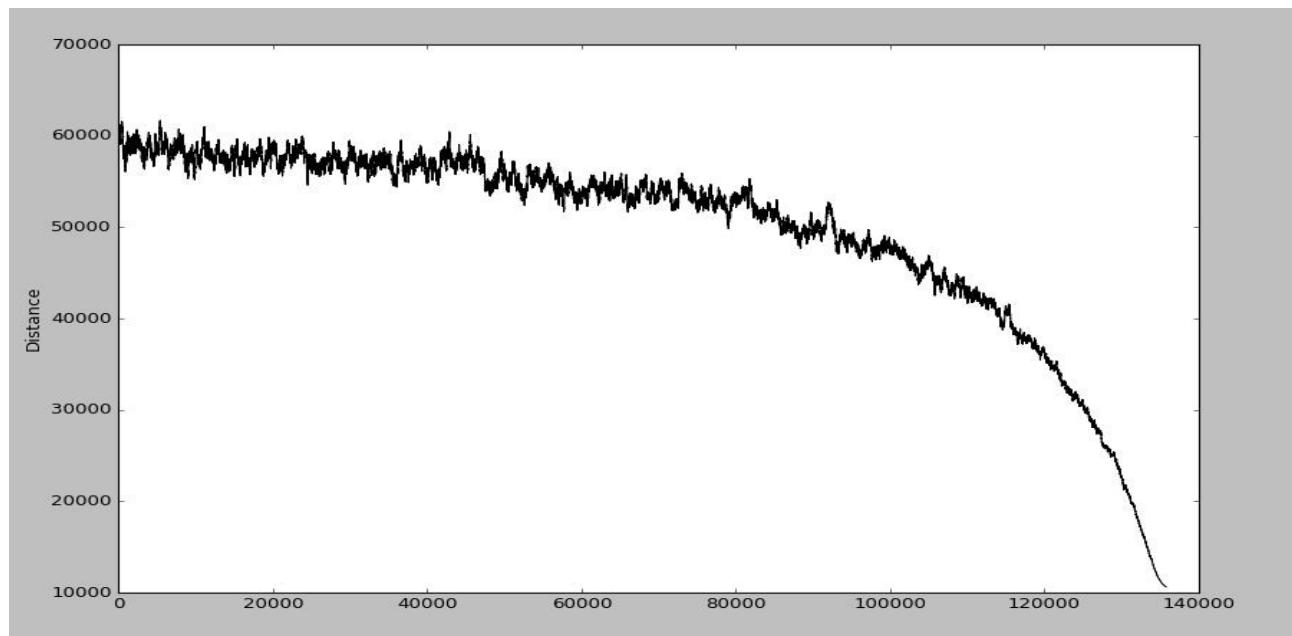
Odczytując wartości z wykresów bez wątpienia można stwierdzić, że lepsza optymalizacja następuje przy użyciu wykładniczej funkcji zmiany temperatury. Dlatego też dla wszystkich wizualizacji z pozostałych podpunktów sprawozdania użyta została właśnie ta funkcja.

c) Poniżej przedstawione wizualizacje przedstawiają działanie funkcji minimalizującej funkcję celu (distance). Algorytm działał dla najbardziej efektywnej konfiguracji, opracowanej na podstawie wniosków z poprzednich podpunktów, tzn. użycie arbitrary swap oraz wykładniczej funkcji zmiany temperatury.

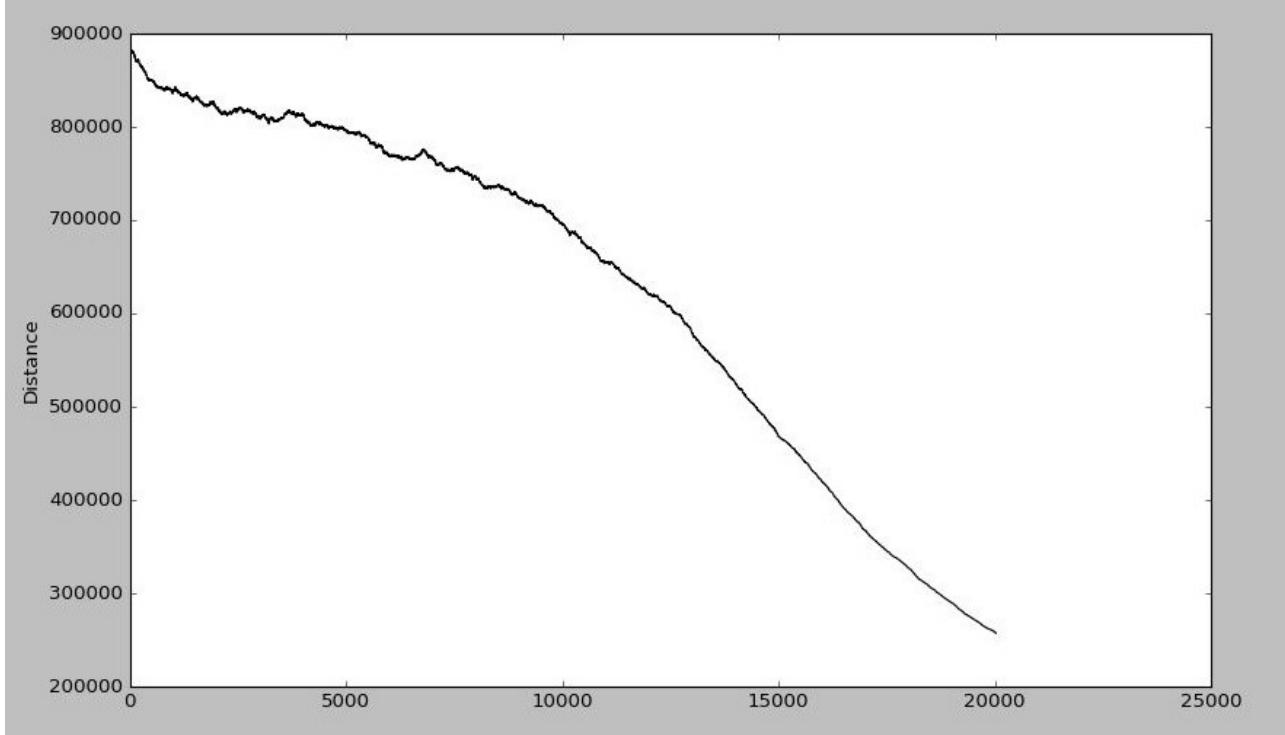
* Rozkład jednostajny, N = 100:



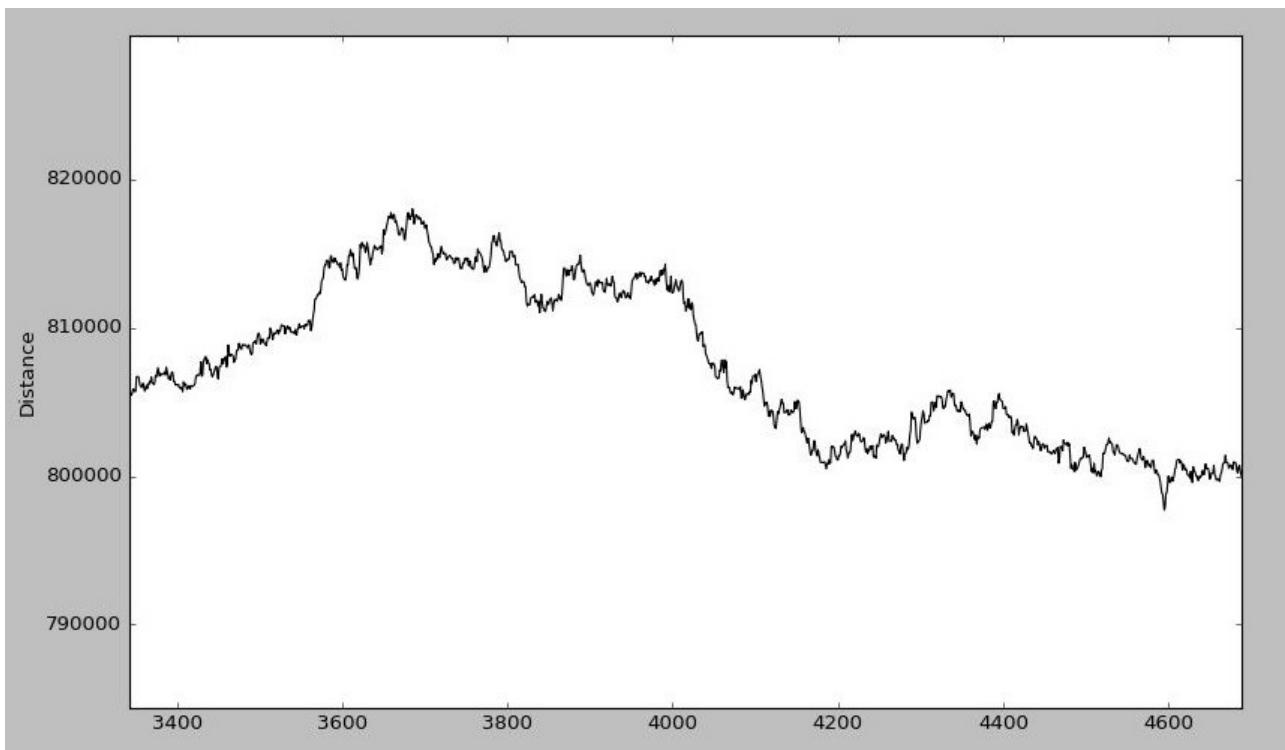
* Rozkład normalny, N = 700



* Rozkład grupowy, N = 3000



Z uwagi na dużą ilość iteracji oraz początkowej wartości funkcji celu, ten sam wykres w przybliżeniu:



Analizując powyższe wizualizacje można stwierdzić, że procedura minimalizująca funkcję celu działa prawidłowo. Pozwala ona czasem przyjąć mniej korzystny stan (większą wartość funkcji celu), aby nie „utknąć” w minimum lokalnym, co jest główną istotą symulowanego wyżarzania.

2. Generowanie obrazu binarnego oraz przekształcenie ułożenia punktów za pomocą symulowanego wyżarzania dla zdefiniowanej funkcji energii.

Poniżej przedstawione są przykładowe obrazy (300x300) złożone z białych oraz czarnych punktów. Po lewej stronie przedstawiony jest losowo wygenerowany obraz, po prawej stronie znajduje się obraz przekształcony przez działanie algorytmu.

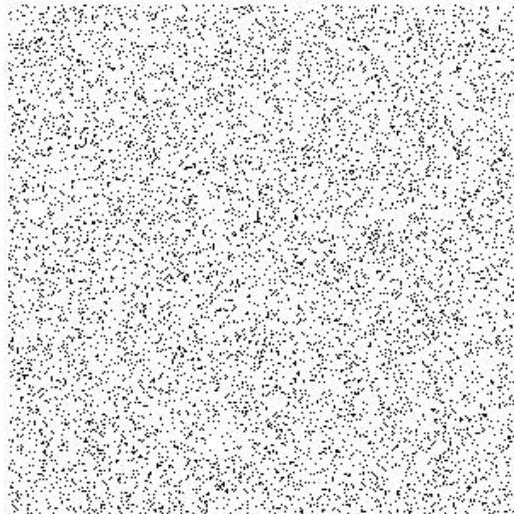
Zdefiniowana energia: czarne punkty w bliskiej odległości (prostokąt o wymiarach 7x7, w którego centrum leży przetwarzany punkt) przyciągają się, zaś białe punkty w identycznym otoczeniu odpychają się z taką samą siłą.

* sąsiedztwo na przekątnych, $\rho = 0,1$, $T_0 = 200$, $I = 0,99999$

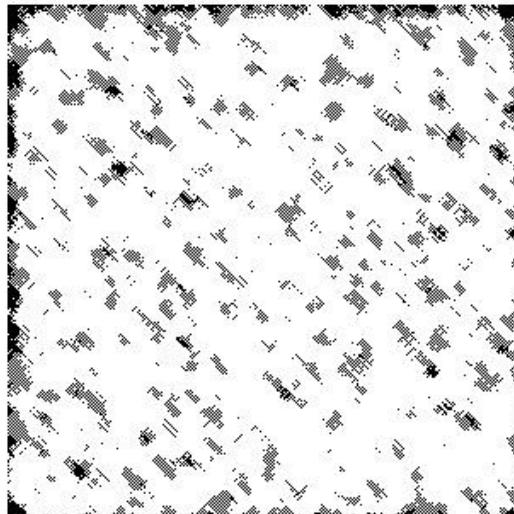
Applet

Total energy: -3.532595E7
Initial temperature: 200.0
Current temperature: 0.009999980350529796

Original image:



Changed image:

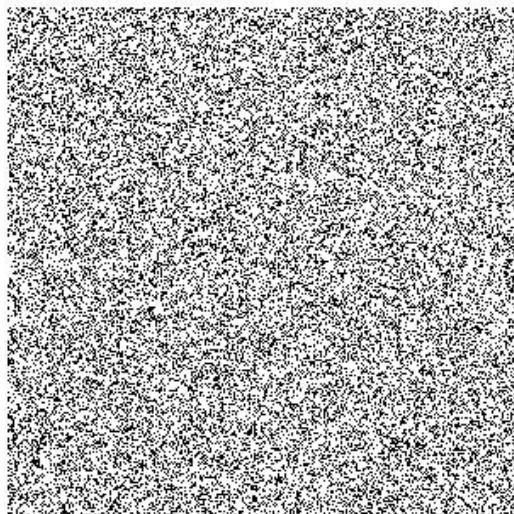


*sąsiedztwo na przekątnych, $\rho = 0,3$, $T_0 = 400$, $I = 0,99999$

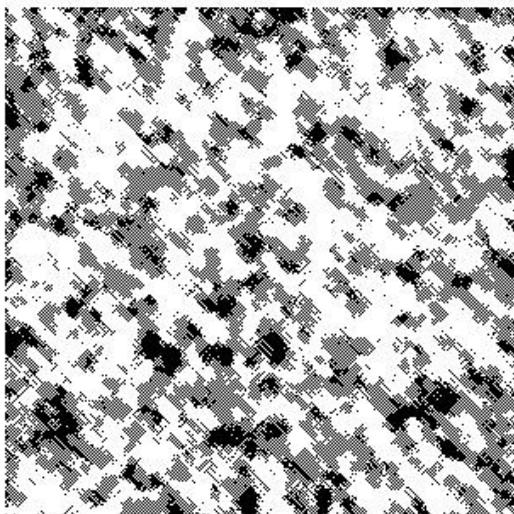
Applet

Total energy: -2.951355E7
Initial temperature: 400.0
Current temperature: 0.009999917498750607

Original image:



Changed image:



* sąsiedztwo na przekątnych, $\rho = 0,4$, $T_0 = 600$, $I = 0,99999$

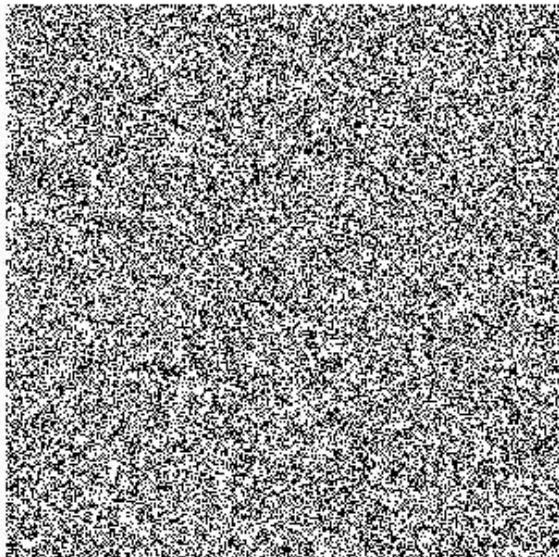
Applet

Total energy: -2.79605E7

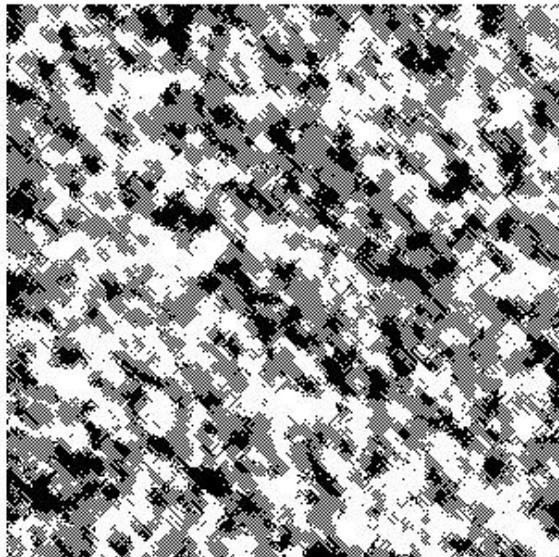
Initial temperature: 600.0

Current temperature: 0.009999948306509441

Original image:



Changed image:



* sąsiedztwo krzyżowe, $\rho = 0,1$, $T_0 = 200$, $I = 0,99999$

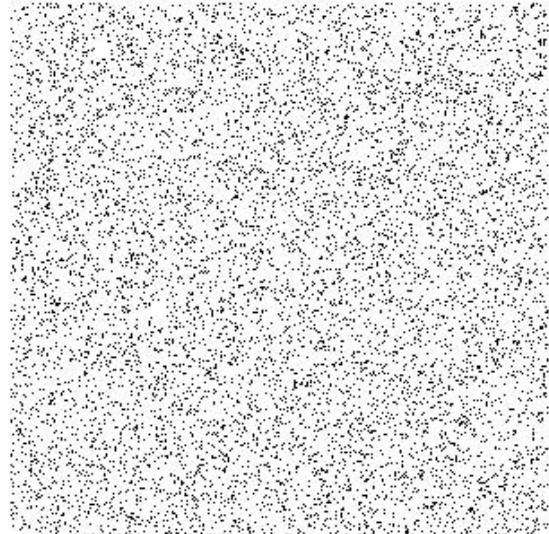
Applet

Total energy: -3.96572E7

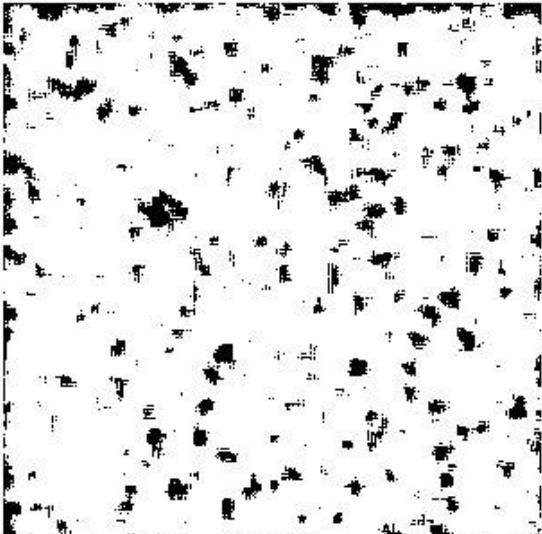
Initial temperature: 200.0

Current temperature: 0.009999980350529796

Original image:



Changed image:



* sąsiedztwo krzyżowe, $\rho = 0,3$, $T_0 = 400$, $I = 0,99999$

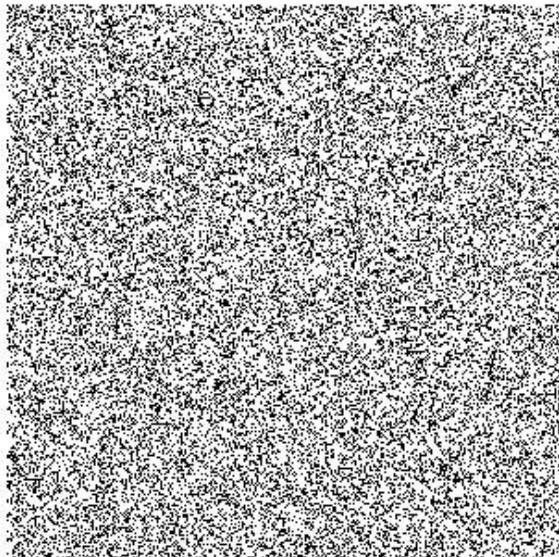
Applet

Total energy: -3.35614E7

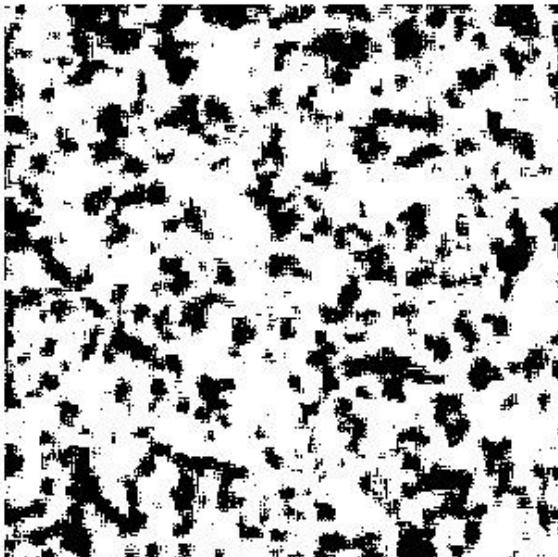
Initial temperature: 400.0

Current temperature: 0.009999917498750607

Original image:



Changed image:



* sąsiedztwo krzyżowe, $\rho = 0,4$, $T_0 = 600$, $I = 0,99999$

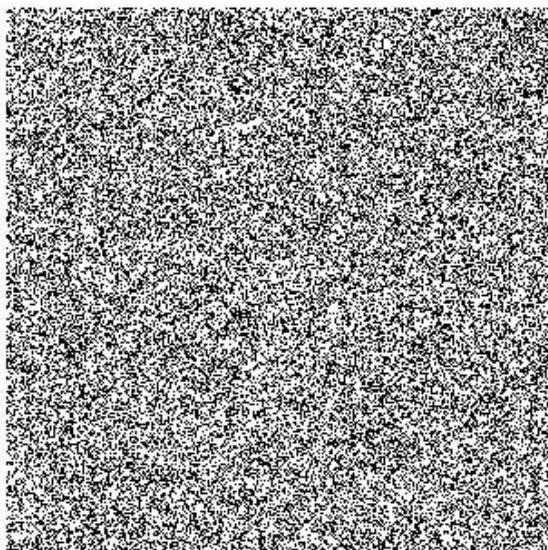
Applet

Total energy: -3.213875E7

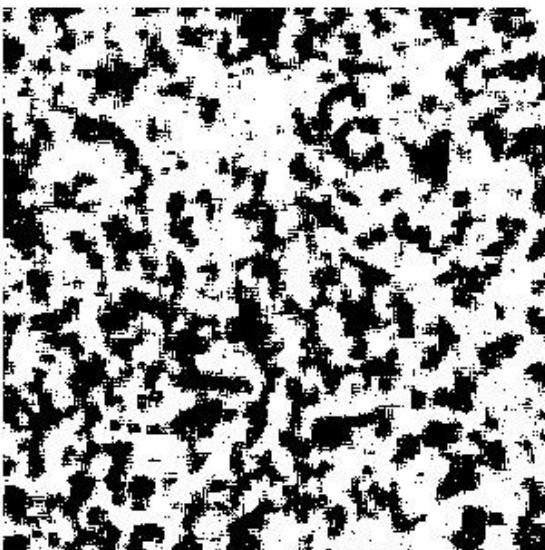
Initial temperature: 600.0

Current temperature: 0.009999948306509441

Original image:



Changed image:



* sąsiedztwo ósemkowe, $\rho = 0,1$, $T_0 = 200$, $I = 0,99999$

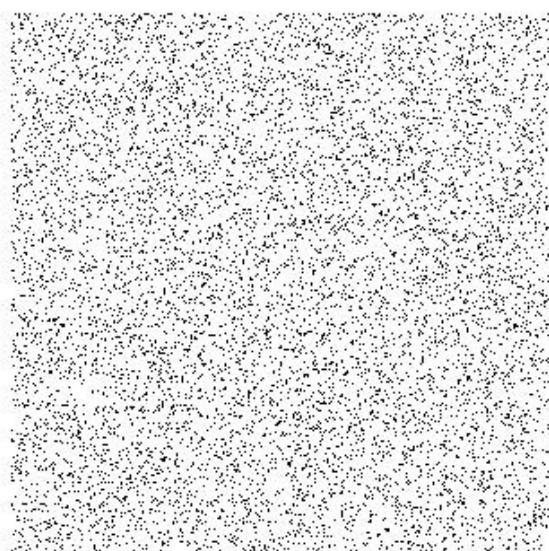
Applet

Total energy: -7.49767E7

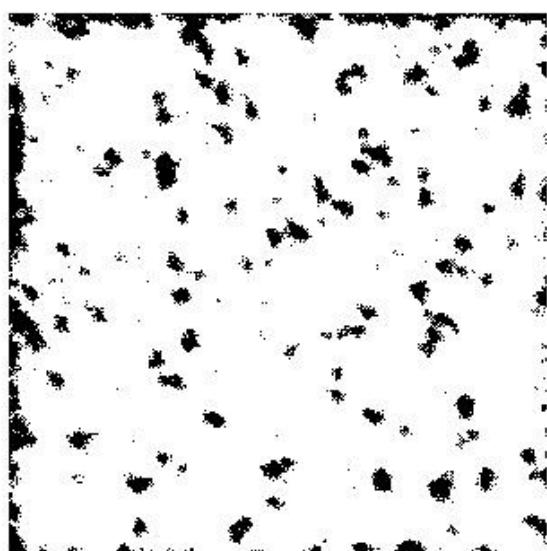
Initial temperature: 200.0

Current temperature: 0.009999980350529796

Original image:



Changed image:



* sąsiedztwo ósemkowe $\rho = 0,3$, $T_0 = 400$, $I = 0,99999$

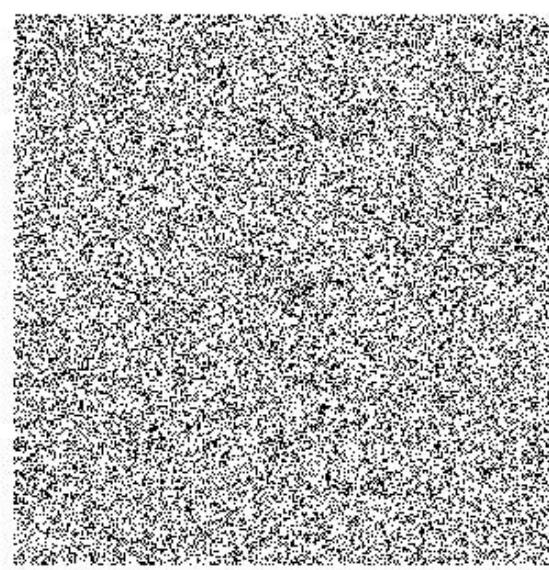
Applet

Total energy: -6.26189E7

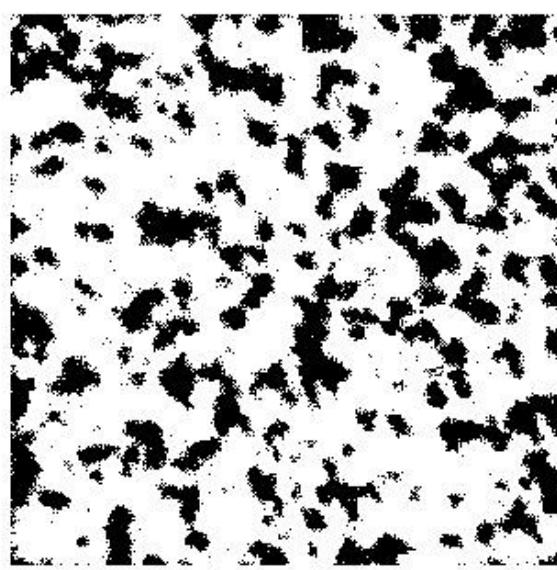
Initial temperature: 400.0

Current temperature: 0.009999917498750607

Original image:



Changed image:



* sąsiedztwo ósemkowe, $\rho = 0,4$, $T_0 = 600$, $I = 0,99999$

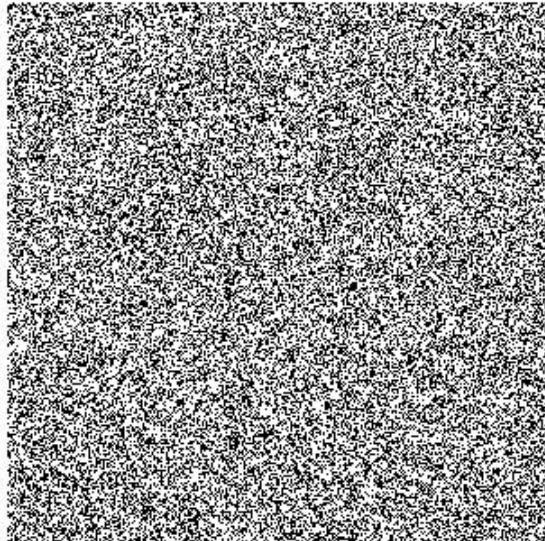
Applet

Total energy: -5.940875E7

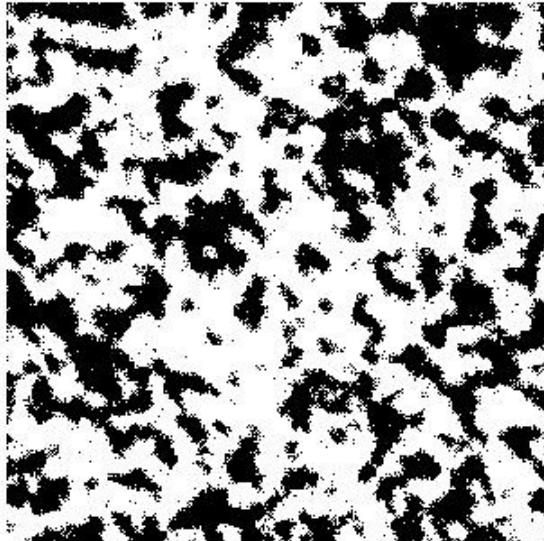
Initial temperature: 600.0

Current temperature: 0.009999948306509441

Original image:



Changed image:



Powyższe wizualizacje powstały poprzez zamianę losowych punktów obrazu, czyli metodą, która okazała się efektywniejsza podczas badania wyników zadania 1. Większa ilość obliczeń w przypadku tego zadania spowodowała potrzebę zmiany więcej niż jednej pary punktów podczas jednej iteracji. W omawianych przypadkach była to zamiana 4 par punktów. Można zaimplementować inny sposób generacji stanów sąsiednich, m.in. omawiany wcześniej consecutive swap, jednak arbitrary w tym przypadku również wydał się bardziej wydajny.

Analizując otrzymane obrazy można zauważać różnice przy zastosowaniu różnych rodzajów sąsiedztwa. Sąsiedztwo na przekątnej oznacza, że sąsiadami są punkty, które znajdują się na jednej z dwóch prostych przechodzących przez punkt ruchem gońca szachowego. Po jego zastosowaniu widać charakterystyczne ułożenie punktów pod kątem około 45 stopni do prostej będącej dolnym bądź górnym obramowaniem obrazu.

Sąsiedztwo krzyżowe oznacza, że sąsiadami są punkty położone w tym samym wierszu lub kolumnie co dany punkt. Skupiska nie są już ustalone pod kątem jak w poprzednim sąsiedztwie, zamiast tego układają się w mniejsze bądź większe krzyżyki.

Ostatnim zaimplementowanym sąsiedztwem jest sąsiedztwo ósemkowe, które łączy obydwa wcześniejsze wspomniane sąsiedztwa. Dzięki temu możemy zaobserwować na wygenerowanych obrazach cechy obydwu wcześniejszych sąsiedztw. Punkty skupiają się w krzyżykach zorientowanych pod kątem około 45 stopni do górnego bądź dolnego obramowania obrazu.

Następnie przedstawione zostały różnice w otrzymanym obrazie, w zależności od szybkości spadku temperatury. Wszystkie obrazy zostały wygenerowane dla przypadku:

* $T_0 = 400$, $\rho = 0,4$, sąsiedztwo krzyżowe

$I = 0.99$

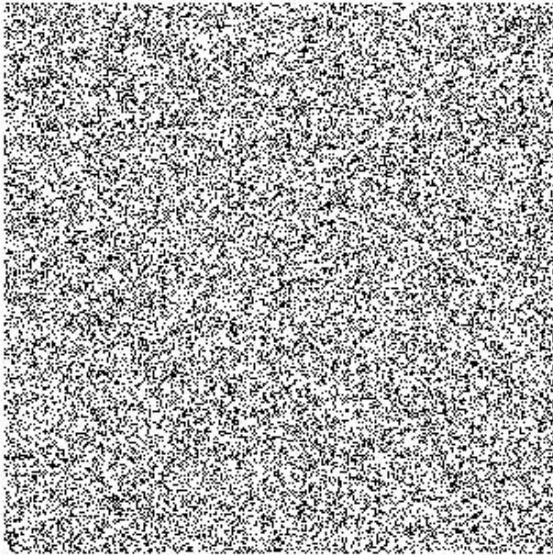
Applet

Total energy: -2.60423E7

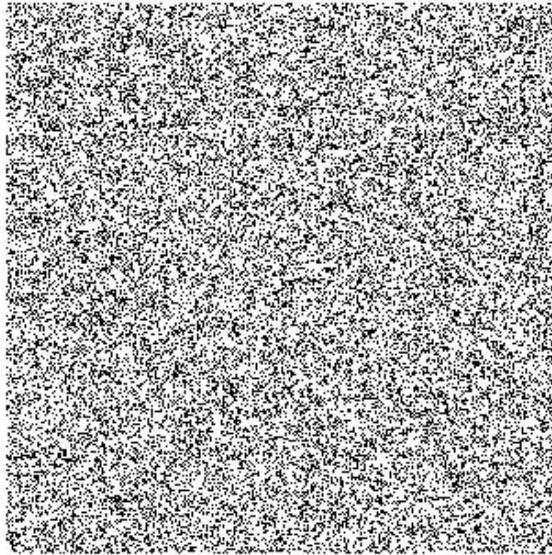
Initial temperature: 400.0

Current temperature: 0.009935512904060476

Original image:



Changed image:



$I = 0.999$

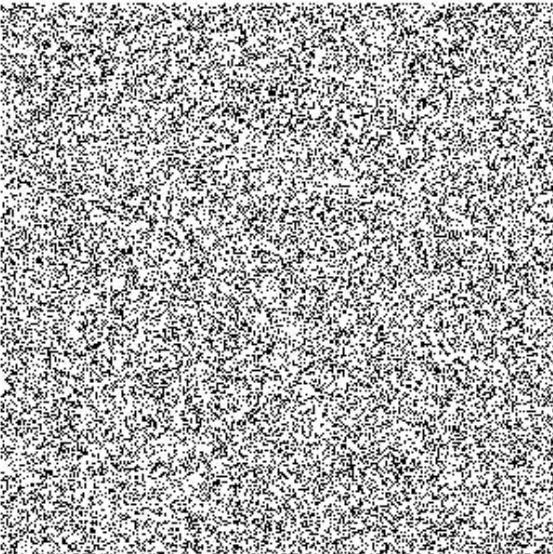
Applet

Total energy: -2.671125E7

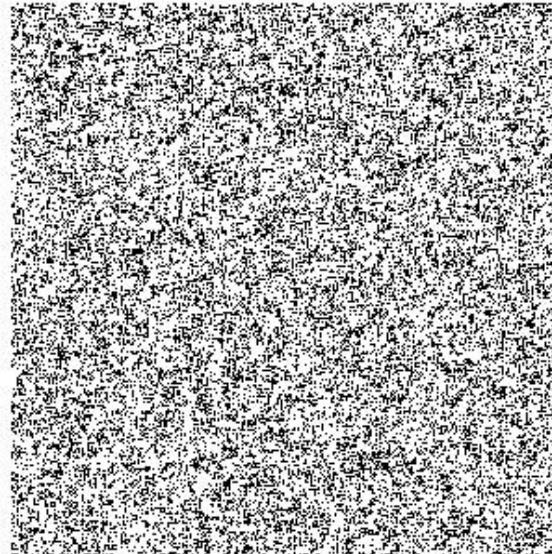
Initial temperature: 400.0

Current temperature: 0.009993354207099823

Original image:



Changed image:



$I = 0.9999$

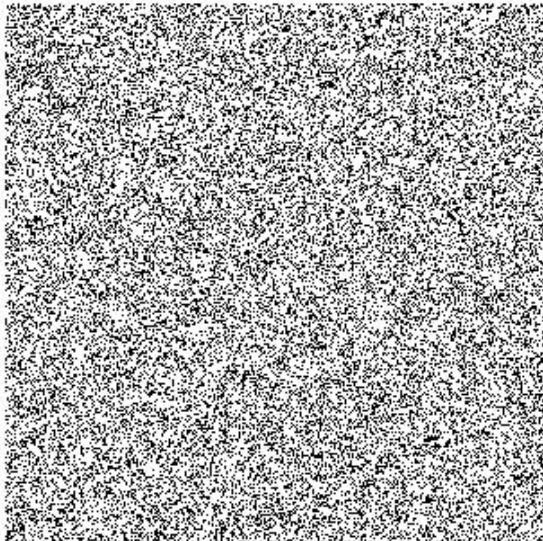
Applet

Total energy: -2.912205E7

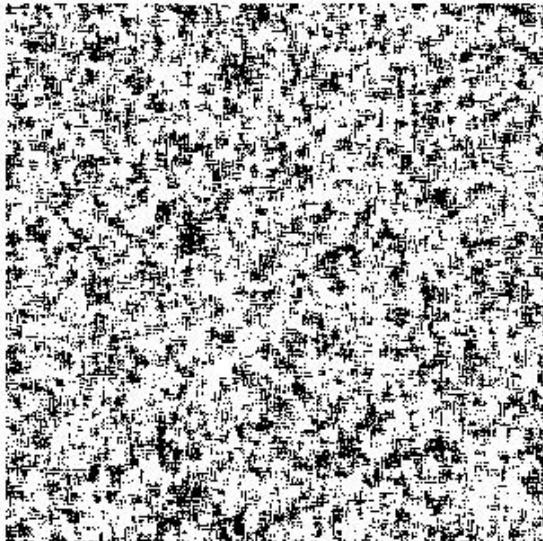
Initial temperature: 400.0

Current temperature: 0.009999048922969498

Original image:



Changed image:



$I = 0.99999$

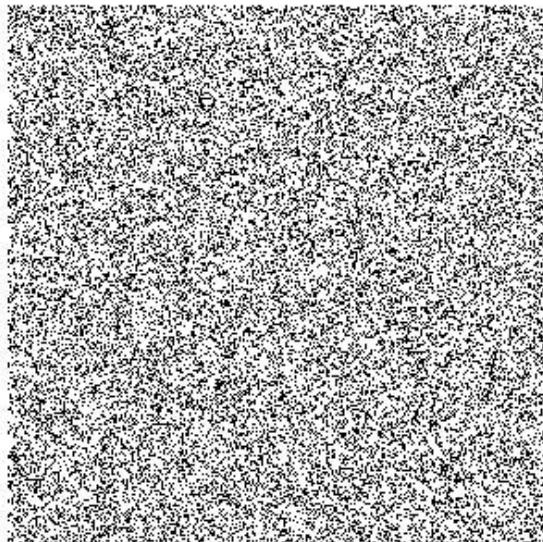
Applet

Total energy: -3.35614E7

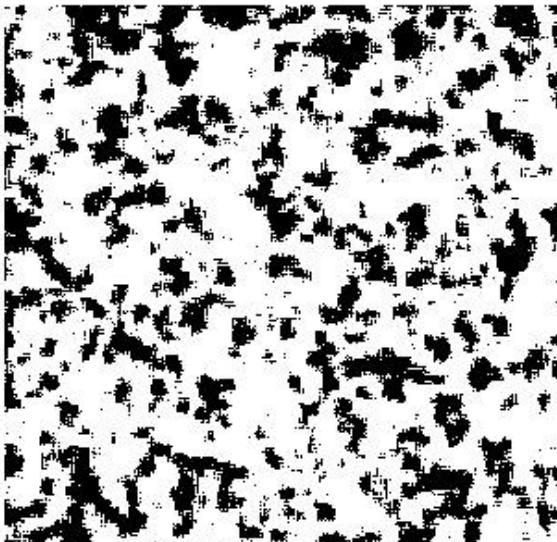
Initial temperature: 400.0

Current temperature: 0.009999917498750607

Original image:



Changed image:



$I = 0.99999$

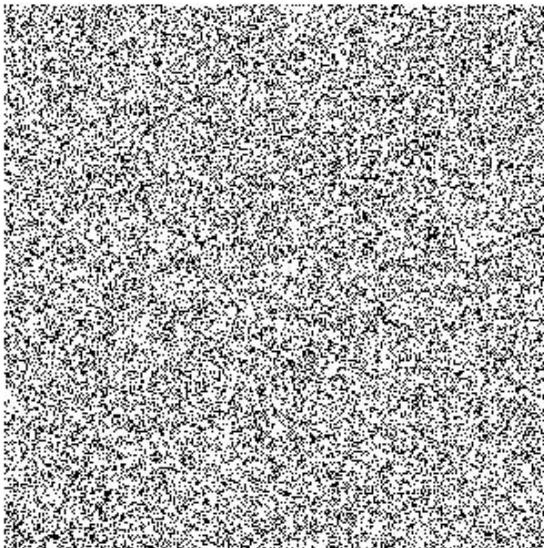
Applet

Total energy: -4.067935E7

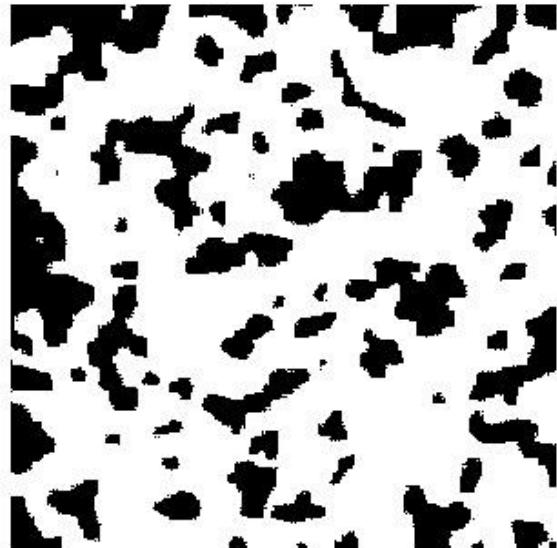
Initial temperature: 400.0

Current temperature: 0.00999999434473098

Original image:



Changed image:



Oczywistym jest, im I jest bliższe 1 tym wolniejszy spadek temperatury. W powyżej przedstawionych przykładach można też zaobserwować, że im spadek temperatury jest wolniejszy tym osiągana jest mniejsza wartość energii, co jest dobrym zjawiskiem, ponieważ celem jest właśnie minimalizacja wartości funkcji energii. Z każdym zwiększeniem I obserwowana jest coraz wyraźniej wyłaniająca się struktura. Zaimplementowana energia polega na przyciąganiu się punktów białych i czarnych, dzięki czemu tworzą się skupiska punktów w poszczególnym kolorze.

Na zmianę generowanych obrazów wpłynie oczywiście zmiana zaimplementowanej energii, oto parę przykładów:

* Pozostawione zostało przyciąganie się punktów czarnych, lecz przyciąganie się białych zamieniono na odpychanie się, z taką samą siłą jak wcześniej się przyciągały:

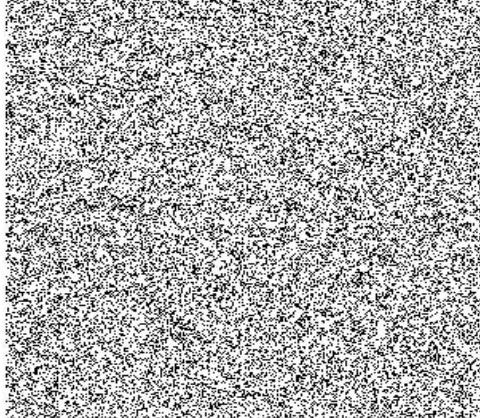
Applet

Total energy: 1.77426E7

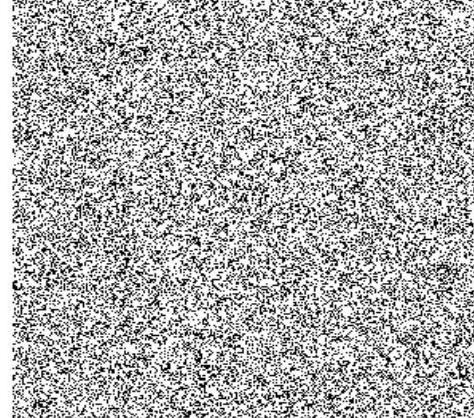
Initial temperature: 400.0

Current temperature: 0.009999917498750607

Original image:



Changed image:



* W tym przypadku energia w ogóle nie dotyczy punktów białych, zaimplementowane zostało jedynie przyciąganie się punktów czarnych.

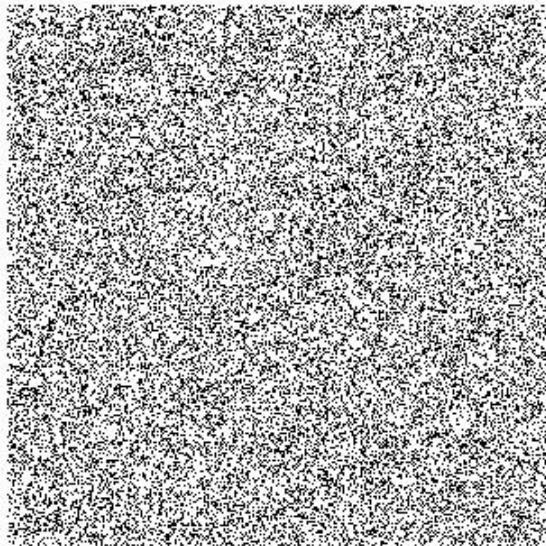
Applet

Total energy: -7943850.0

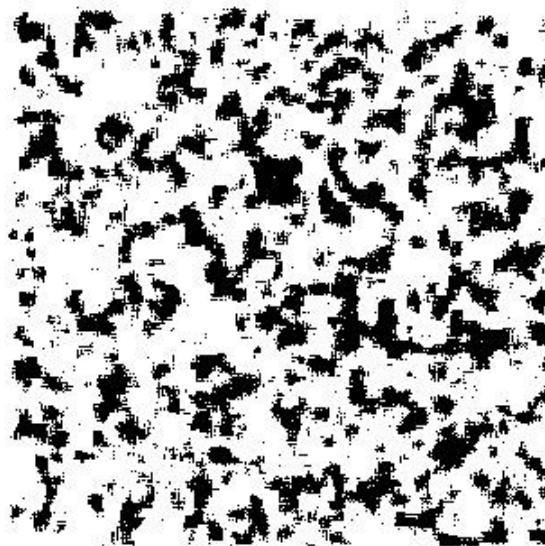
Initial temperature: 400.0

Current temperature: 0.009999917498750607

Original image:



Changed image:



* poniższa wizualizacja przedstawia sytuację, gdy przyciąganie punktów czarnych zachodzi jedynie w jednej płaszczyźnie:

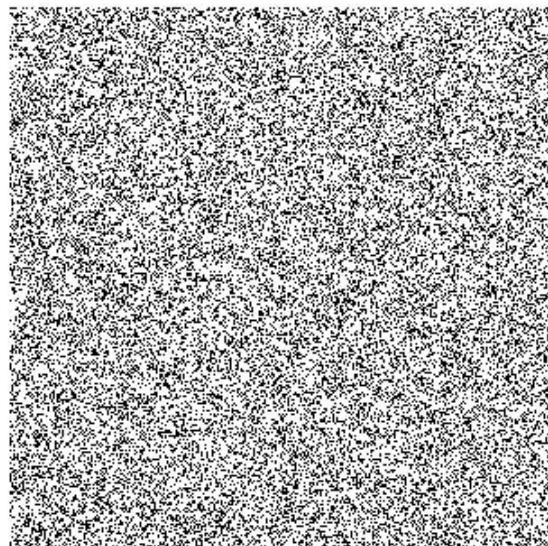
Applet

Total energy: -4279050.0

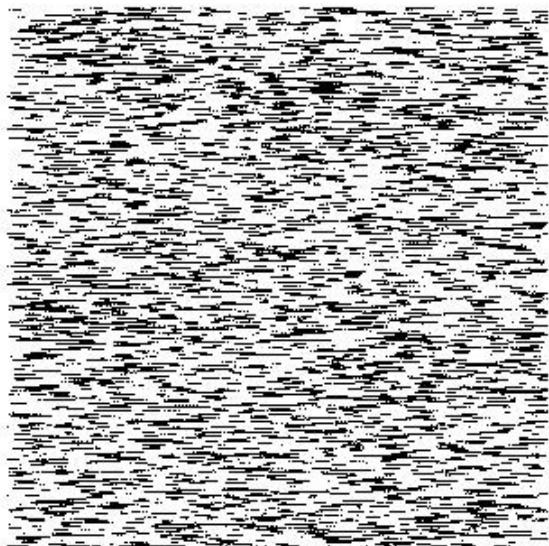
Initial temperature: 400.0

Current temperature: 0.009999917498750607

Original image:



Changed image:



Powyższe wizualizacje pozwalają jednoznacznie stwierdzić, że nawet niewielkie zmiany w sposobie implementacji funkcji energii powodują znaczne zmiany w końcowym wyniku. Zmiana funkcji energii może doprowadzić do generowania bardzo różnych struktur.

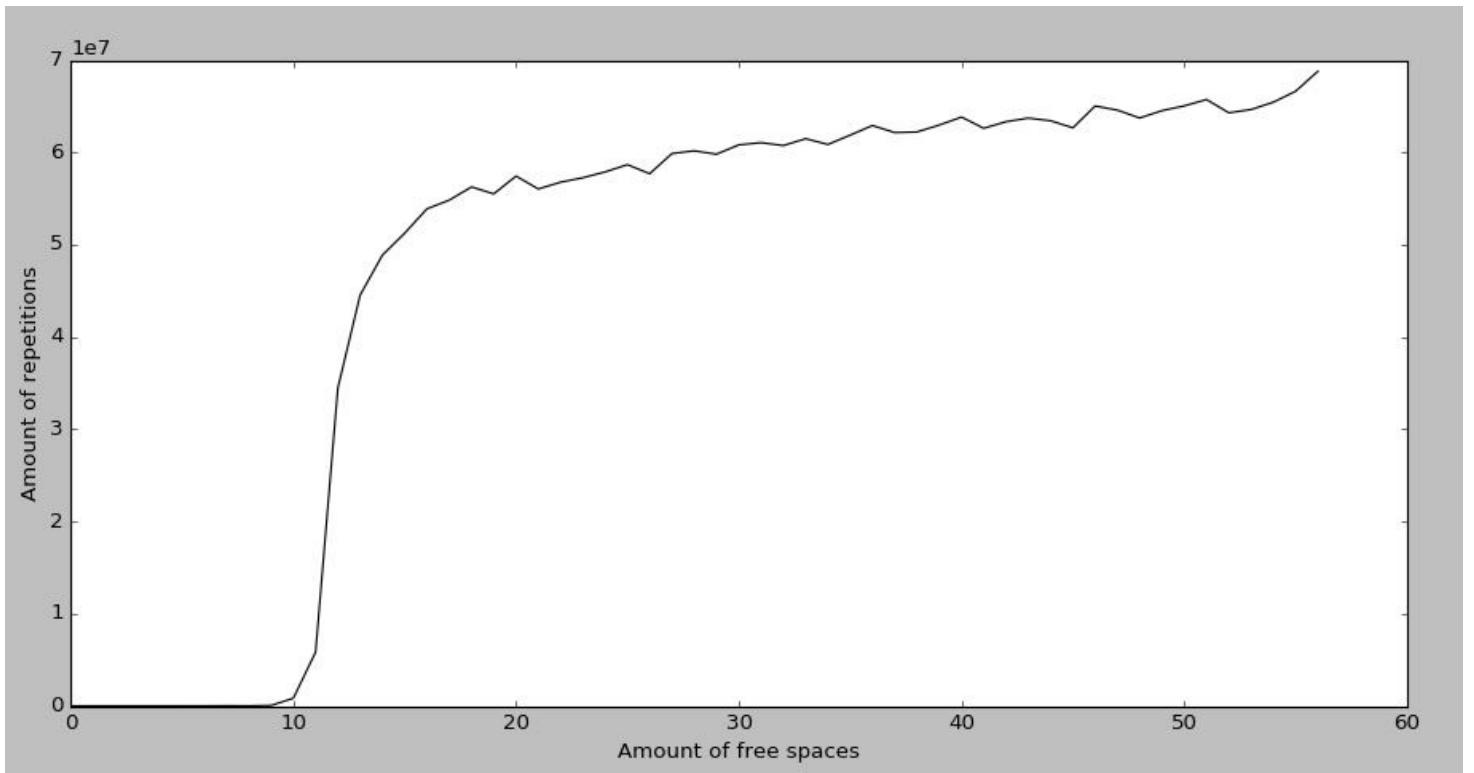
3. Program rozwiązujący sudoku.

Oto wizualizacja działania programu dla przykładowego sudoku, które w serwisie udostępniającym przykładowe łamigłówki zakwalifikowało tą wejściową konfigurację do najtrudniejszej kategorii.

```
<terminated> Sudoku [Java Application] C:\Program Files\Java\jre1.8.0_66\bin\javaw.exe
Starting board:
0 7 0 0 0 0 0 0 0
2 0 4 0 0 9 0 0 0
0 0 9 3 0 0 4 0 0
0 5 0 0 0 0 9 2 0
0 2 0 0 0 5 0 0 7
0 1 0 7 0 0 0 6 0
6 0 0 5 9 4 0 0 0
0 0 0 0 0 0 0 0 2
0 0 1 0 0 7 6 0 4

Starting value: 70
Amount of iterations: 9210335
Solved sudoku:
1 7 3 4 5 6 2 8 9
2 6 4 8 7 9 1 3 5
5 8 9 3 2 1 4 7 6
4 5 7 6 8 3 9 2 1
3 2 6 9 1 5 8 4 7
9 1 8 7 4 2 5 6 3
6 3 2 5 9 4 7 1 8
7 4 5 1 6 8 3 9 2
8 9 1 2 3 7 6 5 4
Final value = 0
```

Po prawej stronie zrzutu widać plik wejściowy zawierający konfigurację początkową. Program rysuje ją zanim rozpoczęcie działania algorytmu wyżarzania zastępując X na 0 i dla lepszej wizualizacji dodając odpowiednie spacje. Następnie tablica sudoku jest zapełniana losowym liczbami naturalnymi z przedziału <1:9>. Starting value to ilość niedozwolonych powtórzeń (liczonych dla kolumn, wierszy oraz podtablic 3x3), jest to funkcja, która będzie minimalizowana. Gdy osiągnie ona wartość 0 wyżarzanie jest przerywane, gdyż oznacza to rozwiązanie łamigłówki. Program rysuje rozwiązanie oraz wypisuje ostateczną ilość powtórzeń. W przypadku, gdyby program nie rozwiązał łamigłówki wartość ta byłaby niezerowa, a rozwiązanie, wyświetlane jako „solved sudoku” wcale nie byłoby prawidłowym. Jednakże dla testowanych, rozwiązywalnych sudoku nie stwierdzono przypadku, którego po dopasowaniu odpowiednich parametrów wyżarzania nie udałoby się rozwiązać. Podanie nierozwiązywalnego sudoku spowoduje, że algorytm przerwie się dopiero po osiągnięciu temperatury minimalnej, a wyświetcone rozwiązanie będzie zawierać możliwie jak najmniej powtórzeń.



Powyższy wykres przedstawia zależność ilości iteracji od ilości pustych miejsc w wprowadzanym sudoku. Zależność ta jest bardzo interesująca ze względu na bardzo nagły wzrost w okolicy fS (freeSpaces) = 10. Dla $fS = 9$ miały miejsce 3674 iteracje, a dla $fS = 13$ już ponad 34 miliony. Na całe szczęście wzrost ten hamuje w okolicy $fS = 15$ i problem ten daje się rozwiązać dla całego sudoku. Największa odczytana wartość wyniosła prawie 70 milionów iteracji dla $fS = 58$, co pozwala na zastosowanie tego algorytmu na średniej jakości sprzęcie w czasie mniejszym niż minuta na jedno wywołanie.