Sprawozdanie

Metody obliczeniowe w nauce I technice

Temat: Symulowane wyżarzanie

Autor: Ryszard Sikora

Kod źródłowy wszystkich zadań dostępny jest pod adresem: https://github.com/rychuhardy/mownit-lab4.git

Zadanie 1: Travelling salesman problem

Doświadczenie przeprowadzono w kilku wariantach, gdzie zmieniane były odpowiednio parametry:

- **n** – liczba wierzchołków
- **it** – liczba iteracji
- **t** – temperatura początkowa

oraz metody rozwiązywania:

- **consecutive swap**/**arbitrary swap**

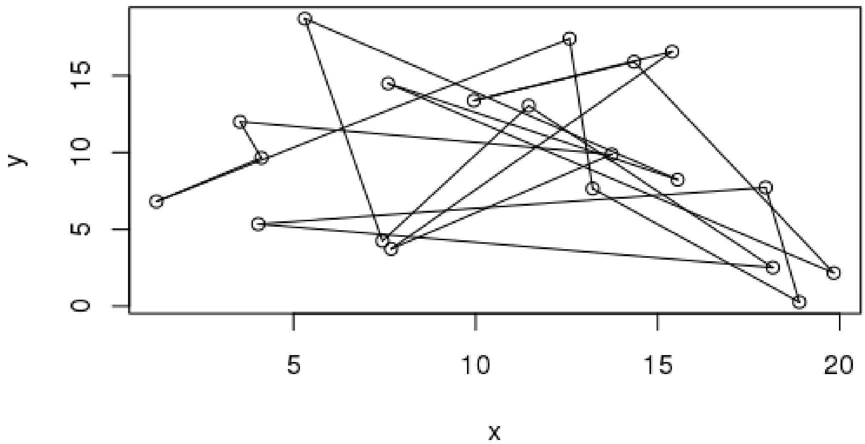
Inne parametry używane w algorytmie, których wartości były stałe:

- **t_factor** – eksponencjalny spadek temperatury*
- **in_it** – liczba iteracji dla tej samej wartości t temperatury (tzn. liczba prób zamian kolejności punktów) o wartości 5
- **t_lowest** – najniższa temperatura o wartości 0.1

*__t_factor__ był obliczany za pomocą parametrów **t**, **t_lowest** oraz **it**, ale w taki sposób, że spadek temperatury był eksponencjalny.

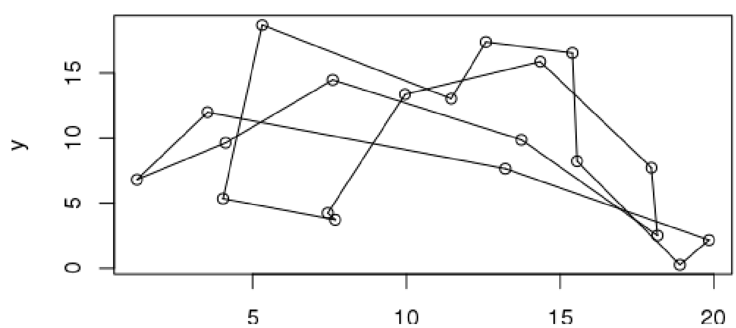
Przykładowe rezultaty dla układu 20 punktów z rozkładu jednostajnego

Początkowa ścieżka dla każdego z niżej prezentowanych rozwiązań

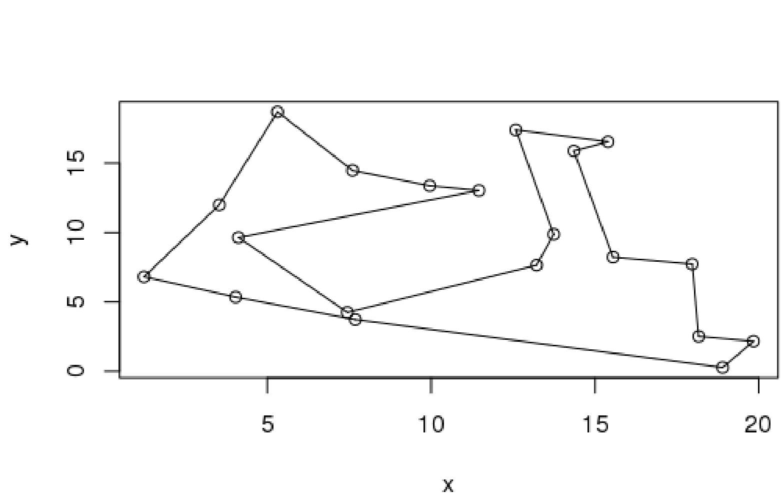


- Przykładowe rozwiązania dla różnych wartości parametru **it**, ze stałą temperaturą początkową **t**=50 oraz metodą **arbitrary swap**

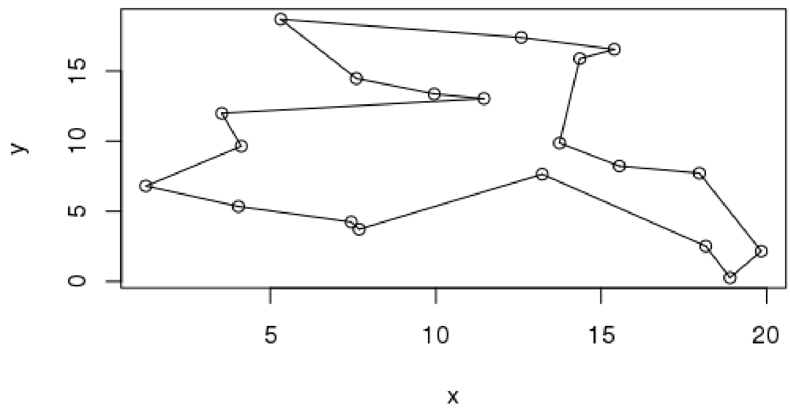
Dla **it**=10



Dla **it**=1000



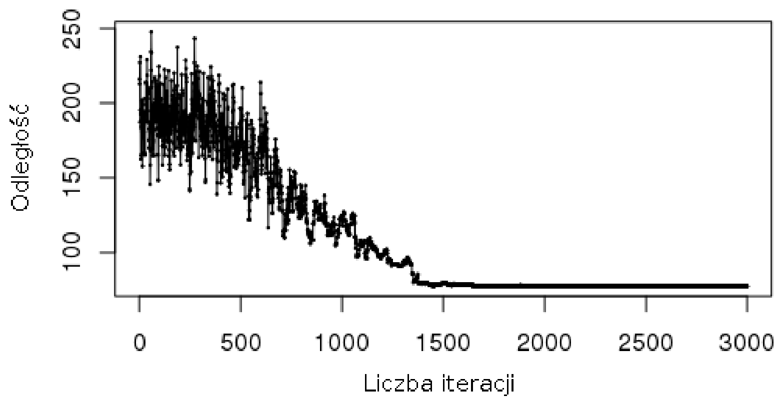
Dla **it**=3000



Zestawienie dokładnych wyników dla wyżej wymienionych przykładów

Liczba iteracji	Odległość
0	216.1601
10	132.6618
1000	94.88699
3000	77.4942

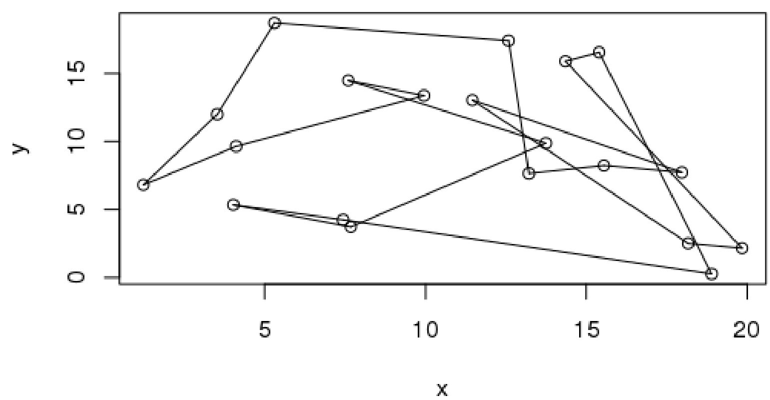
Wykres zmian odległości w kolejnych iteracjach:



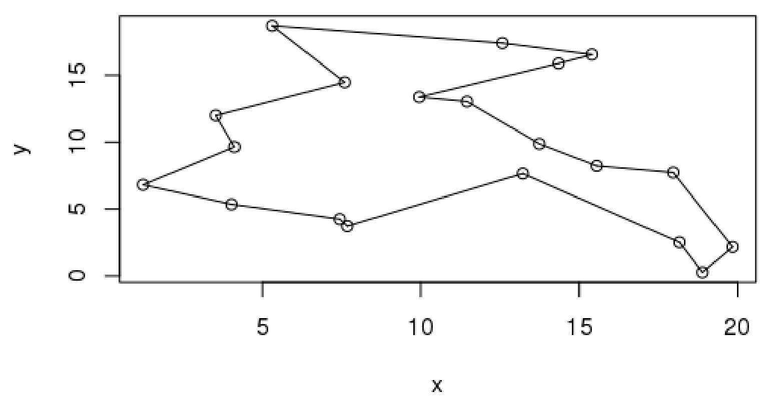
Można zauważyć, że od 1500 iteracji wynik nie poprawiał się.

- Kolejne rozwiązania dla innej temperatury początkowej **t**=100, metodą **arbitrary swap**

Dla **it**=30



Dla **it**=1000



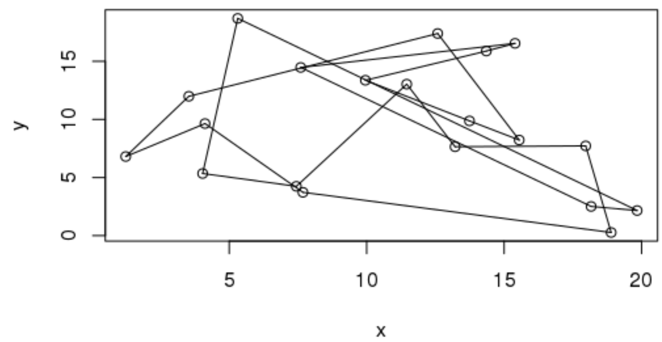
Zestawienie dokładnych wyników dla wyżej wymienionych przykładów:

Liczba iteracji	Odległość
0	216.1601
30	139.4603
1000	74.59016

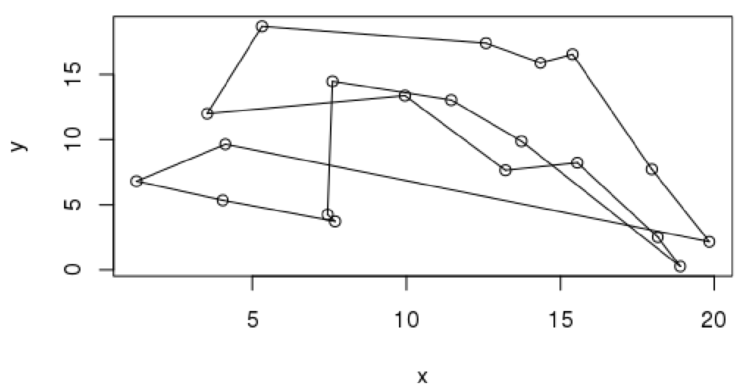
Porównując z wartościami uzyskanymi dla **t**=50 można zauważyć, że dla **t**=100 już po 1000 iteracji uzyskano wynik lepszy niż dla 3000 iteracji z **t**=50.

- Kolejne rozwiązanie wykorzystywało metodę **consecutive swap**, z tą samą temperaturą początkową co poprzednie rozwiązanie **t**=100

Dla **it**=30



Dla **it**=1000

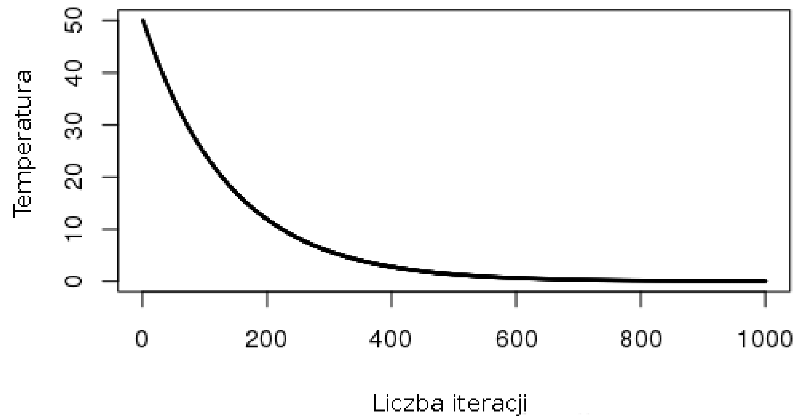


Zestawienie wyników dokładnych dla powyższych przykładów:

Liczba iteracji	Odległość
0	216.1601
30	154.7685
1000	115.4696

Można zauważyć, że metoda wykorzystująca **consecutive swap** uzyskała gorsze wyniki od **arbitrary swap** nawet dla niższej temperatury początkowej.

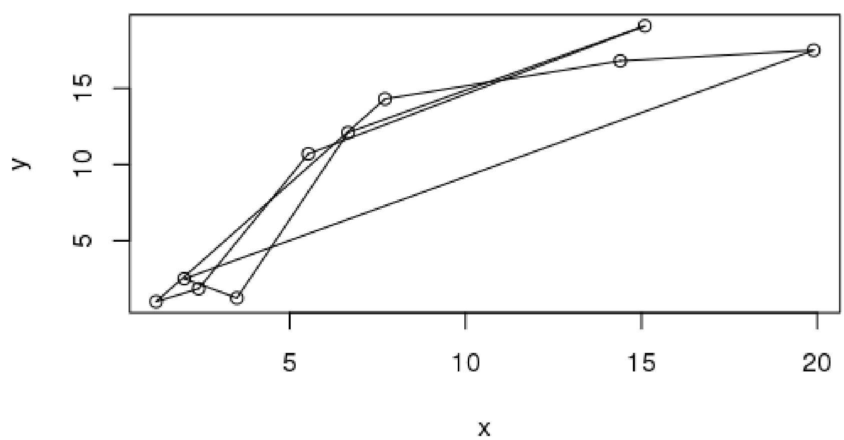
Wykres przedstawiający zmianę temperatury w zależności od liczby iteracji



Kształt wykresu jest taki sam dla każdego z wyżej wymienionych przypadków

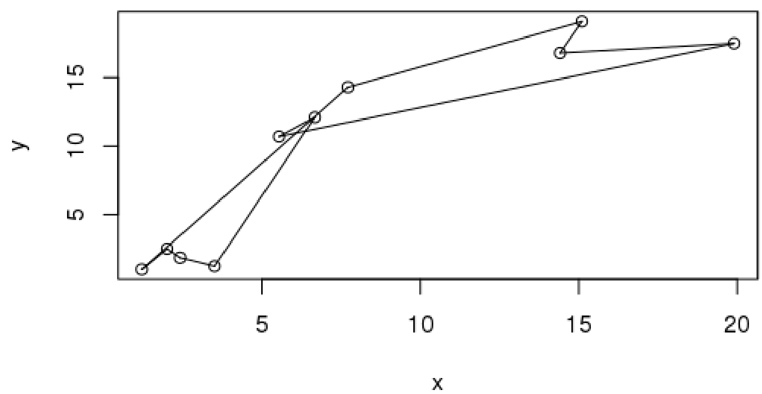
Przykładowe rezultaty dla układu 10 punktów

Początkowy układ punktów

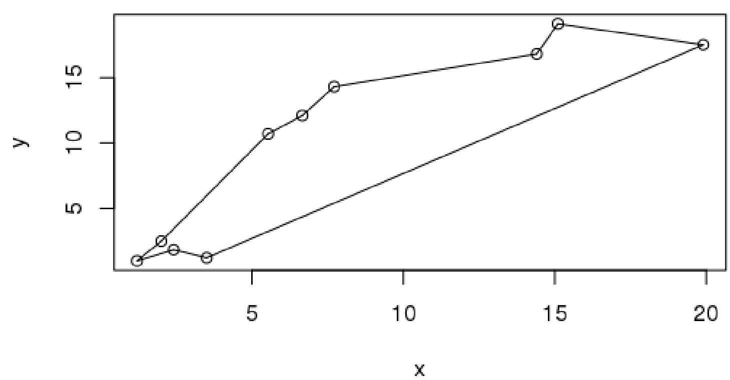


- Rozwiązanie dla **t**=100 oraz **consecutive swap**

Dla **it**=10



Dla **it**=1000



Zestawienie wyników dokładnych dla powyższych przykładów:

Liczba iteracji	Odległość
0	98.68832
10	64.30187
1000	55.29934

W przypadku wierzchołków pochodzących z trzech różnych klastrów rezultaty są podobne do rozkładu jednostajnego.

Zadanie 2: Obraz binarny

W rozwiązaniach zadań 2 i 3 korzystam z funkcji sann z biblioteki ConsPlan, zamiast własnej implementacji, ze względu na szybkość działania.

Wygenerowany obraz o rozmiarze 20x20 i gęstości czarnych punktów 0,6:

Pierwsza testowana funkcja kosztu zwracała wartość bezwzględną różnicy pomiędzy liczbą sąsiadów białych i czarnych. Punkt może mieć 3, 5 lub 8 sąsiadów (pierwsze dwa przypadki dotyczą punktów w rogach i na bokach). Przykładowo jeśli punkt ma 6 białych i 2 czarnych sąsiadów, to niezależnie od koloru tego punktu koszt wyniesie **|6-2|**.

Funkcja generująca kolejne stany losowała punkt ze środku obrazu (czyli taki, który ma 8 sąsiadów) i zmieniała kolor każdego z jego sąsiadów na przeciwny.

Obraz wygenerowany po 30000 iteracji:

Inna funkcja generująca stany sąsiednie ustawiała kolor swoich sąsiadów po bokach na swój własny kolor, a sąsiadów w rogach na przeciwny do swojego.

Wynik po 30000 iteracji:

Druga testowana funkcja kosztu zwracała liczbę sąsiadów o kolorze przeciwnym niż kolor danego punktu.

Funkcja generująca stany sąsiednie tym razem zmieniała kolor sąsiadów w lewej dolnej części na czarny, a w prawej górnej części na biały.

Efekt uzyskany po 30000 iteracji:

Zadanie 3: Sudoku

Plansze z testowanymi sudoku znajdują się w plikach tekstowych w repozytorium. Sudoku 1 to (według niektórych) najtrudniejsze znane do tej pory sudoku. Sudoku 2 jest prostym sudoku rozwiązywalnym za pomocą prostej dedukcji. Sudoku 3 ma poziom 'diabelski'.

Zastosowano dwa podejścia do rozwiązania problemu:

- Początkowo wypełniając sudoku losowymi liczbami z puli możliwych liczb. Funkcje generujące stany sąsiednie zamieniały dwa losowe elementy w sudoku.
- Wpisując zera w miejsca nieuzupełnione. Funkcje generujące stany sąsiednie wpisywały w losowe miejsca liczby z dostępnej puli.

Funkcja kosztu zwracała sumę liczby duplikatów w każdym wierszu, kolumnie i kwadracie 3x3.

Podejście 1

Pierwsza funkcja generująca stany sąsiednie zamieniała dwa losowo wybrane elementy z całego sudoku. Dla tej funkcji algorytm najczęściej zwracał sudoku 2 z kosztem około 22. Przy bardzo dużej liczbie iteracji (~5 mln) udawało się zredukować koszt do 16.

Druga funkcja przyjmowała pewien niezmiennik ustalany przy początkowym wypełnianiu sudoku. Liczby były wstawiane do kolejnych wierszy z puli liczb dla tego wiersza, a nie do całego sudoku z jednej puli liczb. Dzięki temu liczba duplikatów w wierszach zawsze wynosiła 0. Generowanie kolejnych stanów polegało na zamianie dwóch losowych elementów w losowo wybranym wierszu. W tym wariancie udało się rozwiązać sudoku nr 2, a dla sudoku 1 i 3 otrzymywano koszt mniejszy od 5.

Podejście 2

W tym podejściu funkcja kosztu dodatkowo uwzględniała liczbę zer na planszy, aby wyeliminować je z końcowych rozwiązań.

Pierwsza funkcja generująca stan sąsiedni wpisywała w losowe miejsce (także w miejsca już uzupełnione) wartość z puli. Dla sudoku2 średni wynik to 10.

Kolejna funkcja generująca stan sąsiedni losowała jedno puste pole z sudoku i próbowała wpisać w nie losowo wybraną liczbę z puli możliwych liczb dla tego pola. Jeśli nie dało się wpisać w pole żadnej liczby to funkcja zwracała początkową planszę. Ta funkcja miała najgorsze rezultaty, w okolicach 50.

Przy porównywaniu końcowych wyników podejście 1 ma naturalnie niższe wyniki ze względu na brak zer, które znacząco zwiększają początkowy koszt.