Dokumentacja Projektu	
Temat: **Planowanie trasy łazika marsjańskiego**	
Autorzy:	Marcin Brzózka Stanisław Dudiak

Spis treści

1. Model zagadnienia - opis słowny i matematyczny

Celem projektu jest stworzenia aplikacji umożliwiającej dobór jak najlepszej trasy dla łazika marsjańskiego. Najlepsza trasa to taka, która umożliwi przemieszczenie się łazika z punktu A do B w najkrótszym możliwym czasie. Wymaganymi danymi do zaprojektowania trasy są dwie mapy, które w dyskretny sposób (każdy piksel z osobna) opisują wysokości terenu oraz jakość terenu (im większa wartość mapy jakości terenu, tym większe trudności w jego pokonaniu – większe koszta energetyczne oraz czasowe).

Łazik marsjański ma również pewną określoną pojemność akumulatora. Z tego powodu algorytm nie może decydować się jedynie na taką trasę, która umożliwi najkrótszą podróż, ale musi również zapewnić nierozładowanie się łazika po drodze. W czasie doboru pojemności należy również uwzględnić fakt, że dobrana wartość może być zbyt mała, by możliwe było znalezienie jakiejkolwiek trasy.

Łazik ma zamontowane panele słoneczne umożliwiające ładowanie się, gdy ten porusza się w stronę słońca. W naszym projekcie pozycja słońca została stale przypisana do pozycji NE względem mapy. Dlatego poruszanie się bezpośrednio w tym kierunku umożliwia ładowanie w największym stopniu, natomiast poruszanie się w kierunku N lub E umożliwia ładowanie w mniejszym stopniu. W innych kierunkach nie mamy do czynienia z ładowaniem.

Optymalizacji podlega całkowity czas przemieszczania się między punktami początkowym i końcowym.

$$\min\,(t_c)$$

gdzie czas całkowity jest sumą czasów przejazdów między poszczególnymi punktami dyskretnymi map terenu. Przejazdy mogą odbywać się między sąsiednimi punktami dyskretnymi pionowymi, poziomymi lub na ukos.

$$t_c = t_1 + \cdots + t_n$$

Czas i-tego przejazdu określany jest na podstawie poniższego wzoru:

$$t_i = HCT * |h_i - h_{i-1}| + \frac{1}{2} TCT * (p_i + p_{i-1})$$

$$if\ (h_i < h_{i-1}):\ \ HCT = 0.6\ HCT$$

gdzie:

HCT *(height constant time)* – stała związania z wyliczaniem czasu związanego z mapą wysokości terenu, gdy łazik porusza się w dół przyjmuje 0.6 podstawowej wartości;

TCT *(terrain constant time)* – stała związania z wyliczaniem czasu związanego z mapą jakości terenu;

$h_i,\ h_{i-1}$ – wartości punktów trasy z mapy wysokości terenu;

$p_i,\ p_{i-1}$ – wartości punktów trasy z mapy wysokości terenu;

Zużyta energia i-tego przejazdu określana jest na podstawie poniższego wzoru:

$$e_i = HCE * (h_i - h_{i-1}) + \frac{1}{2} TCE * (p_i + p_{i-1}) - SUN_i$$

$$if\ (h_i < h_{i-1}):\ HCE = 0.5\ HCE$$

$$SUN = \begin{cases} SN_1, & (domyślnie\ 8), \quad if\ x_{i-1} < x_i\ and\ y_{i-1} < y_i \\ SN_2, & (domyślnie\ 4), \quad if\ x_{i-1} < x_i\ or\ y_{i-1} < y_i \\ 0, & inne\ przypadki \end{cases}$$

gdzie:

HCE *(height constant energy)* – stała związania z wyliczaniem energii związanego z mapą wysokości terenu, gdy łazik porusza się w dół przyjmuje 0.5 podstawowej wartości;

TCE *(terrain constant energy)* – stała związania z wyliczaniem energii związanego z mapą jakości terenu;

SUN – zbiór stałych związanych z ładowaniem się łazika w zależności od kierunku jazdy miedzy poszczególnymi punktami dyskretnymi

$h_i,\ h_{i-1}$ – wartości punktów trasy z mapy wysokości terenu;

$p_i,\ p_{i-1}$ – wartości punktów trasy z mapy wysokości terenu;

Energia i-tego przejazdu wykorzystywana jest do obliczania bieżącej posiadanej energii na podstawie wzoru:

$$E_i = E_{i-1} - e_i, \qquad dla\ i = 1, \dots, n$$

gdzie:

E_i, E_{i-1} – posiadana energia po poszczególnych przejazdach;

$E_0 = E_{max}$ – początkowa zadana energia łazika

W przypadku, gdy jakikolwiek $E_i < 0$, trasa nie może zostać wybrana.

2. Algorytm

2.1. Pseudokod algorytmu:

- *Wygenerowanie losowego rozwiązania spełniającego wymagania energetyczne.*
- *Utworzenie dwóch list Tabu – dla całych rozwiązań i dla wybranych punktów dla stałe określone kadencje.*
- *Wykonywanie w pętli następujących operacji, aż zostanie spełniony jeden z warunków zakończenia:*
  - *Utwórz listę sąsiadów na podstawie listy możliwych sposób poszukiwania sąsiadów. Zmieniaj rozwiązanie w tym celu dla 3 metod poszukiwania sąsiedztwa wokół najgorszego punktu, a co pewien ustalony czas wokół losowego punktu.*
  - *Wybierz najlepszego sąsiada, jeśli nie jest na listach Tabu.*
  - *Jeśli nie znaleziono żadnego możliwego sąsiada wybierz najlepsze rozwiązanie z listy Tabu dla całych rozwiązań (zawsze aktywne kryterium aspiracji).*
  - *Sprawdź czy rozwiązanie nie jest lepsze, niż obecnie najlepsze – jeśli tak to je podmień.*
  - *Jeśli rozwiązanie nie jest lepsze od najlepszego przez pewną już liczbę iteracji, to wybierz najlepsze z listy Tabu (włączane kryterium aspiracji).*
  - *Dodaj nowe rozwiązanie do list Tabu na określoną kadencję.*
  - *Dla każdego elementu z list Tabu zmniejsz okres pozostawania na liście i gdy kadencja równa się 0 to usuń element z listy*
- *Zwróć pierwsze i najlepsze rozwiązanie*

Kryterium aspiracji – aktywowanie kryterium aspiracji w czasie wykonywania się algorytmu powoduje podstawienie pod najlepsze aktualne rozwiązanie w danej iteracji najlepszego rozwiązania z pierwszej listy Tabu (z całymi rozwiązaniami). Kryterium jest aktywowane gdy:

- żaden z sąsiadów nie może zostać rozwiązaniem (występują na liście Tabu) lub gdy nie można było go znaleźć ze względu na wymagania energetyczne.
- po pewnej zadanej liczbie iteracji najlepsze rozwiązanie się nie zmieniło – możliwość wyłączanie tego warunku poprzez ustawienie zera dla jego parametru.

Pierwsza lista Tabu (lista na całe rozwiązania) – na tej liście odkładane są całe rozwiązania. Gdy jakiś sąsiad aktualnego rozwiązania jest taki sam jak jakiekolwiek rozwiązanie z tej listy Tabu, to nie może zostać wybrany. Kadencja istnienia danego rozwiązania na tej liście jest możliwa do ustawania, domyślnie równa 30 iteracji.

Druga lista Tabu (lista punktów) – Ze względu małe prawdopodobieństwo wybrania identycznej ścieżki dwukrotnie, zaimplementowano dodatkową listę Tabu. W każdej iteracji algorytmu, z obecnie wybranego rozwiązania losowana jest pewna ilość punktów i dodawana do drugiej listy tabu. W kolejnych iteracjach żadne z rozwiązań sąsiednich nie może zawierać punktów z tej listy. Kadencja każdego punktu jest również ustawiona domyślnie na 30 iteracji.

Warunek zakończenia – podstawowym warunkiem zakończenia jest uzyskanie zadanej liczby iteracji. Ale w celu usprawnienia algorytmu przy dużej ilości iteracji, został dodany dodatkowy warunek zakończenia głównej pętli. Jeżeli przez zadaną liczbę iteracji nie zostanie znalezione lepsze rozwiązanie, algorytm skończy się, pomimo nie wykonania określonej liczby iteracji.

2.2. Opisy opracowanych elementów:

2.2.1. Funkcja generująca mapy (Generate_Matrices) – funkcja została utworzona, aby móc losowo generować mapy wysokości i jakości terenu i tym samym sprawdzić działanie algorytmu dla różnych przypadków. Obie mapy mają postać macierzy.
Mapa wysokości terenu (Height_Matrix_Generate) jest tworzona następująco: pierw losowane są współrzędne punktów, następnie na podstawie punktów tworzone są wzniesienia przy użyciu funkcji Quartic kernel. Na końcu utworzona macierz zostaje przeskalowana.
Mapa jakości terenu (Terrain_Matrix_Generate) jest tworzona następująco: pierw cała mapa pokrywana jest jednakowym terenem (takie same wartości wszystkich elementów macierzy), następnie w pętli losowane są dwie wartości – wartość oraz rozmiar terenu dla tej wartości – z odpowiednich list oraz nanoszone zostają losowo na mapę, mogą pokrywać wcześniejsze wartości.

2.2.2. Funkcja generująca pseudolosowe rozwiązanie między punktem A i B (Find_Random_Solution) – funkcja ta jest wykorzystywania przy szukaniu pierwszego rozwiązania oraz w dwóch sposobach poszukiwania sąsiadów. Funkcja działa w następujący sposób:

- Tworzona jest lista na punkty tworzące trasę łazika i dodawany jest do niej pierwszy punkt A
- Tworzony jest punkt z aktualną pozycją (na początku jest to punkt A) i dopóki temu punktowi nie zostaną przypisane współrzędne punktu B w pętli przypisywana jest mu następna pozycja na mapie umożliwiająca zbliżenie się do punktu B przez wylosowanie kierunku (poziomo, pionowo lub na ukos) oraz dodawany jest ten punkt do listy rozwiązania.

- ❖ Są pewne ograniczenia, gdy na przykład pierwszy punkt miał pozycję (0,0), a końcowy (2,9), to po osiągnięciu przez aktualny punkt pozycji x = 2, nie zostanie wartość tej współrzędnej już przekroczona
- ❖ Na końcu zwracana jest lista z punktami tworzącymi rozwiązanie.

2.2.3. Klasa TabuSearch – umożliwia zebranie wszystkich wymaganych parametrów i danych w jednym obiekcie, dzięki czemu prostsze jest ich późniejsze wykorzystywanie – nie trzeba za każdym razem podawać ich jako parametry kolejnych funkcji. Atrybutami klasy są macierze wysokości i jakości terenu, rozmiar mapy, punkt początkowy i końcowy, energia początkowa łazika, parametry HCT, HCE, TCT, TCE, rozwiązania – pierwsze, aktualne, najlepsze, listy Tabu, lista wybranych metod poszukiwania sąsiadów, kadencje list tabu (kT1, kT2), zmienna częstotliwości aktywowania 2 kryterium aspiracji oraz częstotliwość wyboru losowego punktu zamiast najgorszego przy tworzeniu otoczenia. Poniżej opisane są metody tej klasy.

2.2.4. Metoda poszukująca pierwsze rozwiązanie (first_solution) – umożliwia znalezienie pierwszego rozwiązania będącego troszkę bardziej losowego w porównaniu z uzyskanym tylko z funkcji Find_Random_Solution oraz spełniającego warunki energetyczne. Funkcja losuje dwa punkty pośrednie i między nimi oraz początkowym i końcowym tworzy trasę z wykorzystanie funkcji Find_Random_Solution. Po utworzeniu trasy sprawdzane jest, czy łazik po drodze się nie rozładuje. Jeśli tak, to procedura wykonywana jest od początku. Po zbyt dużej liczbie nieudanych prób zwracana jest informacja, że nie można znaleźć rozwiązania dla tak niskiej wartości energii początkowej łazika.

2.2.5. Metody obliczania czasu i zużytej energii na przejeździe między dwoma sąsiednimi punktami rozwiązania (calculate_time, calculate_energy) oraz obliczające całkowity czas dla rozwiązania (calculate_total_time) oraz sprawdzająca, czy łazik się nie rozładuje (try_energy) – działają dokładnie tak, jak to zostało opisane w opisie problemu i modelu matematycznym.

2.2.6. Metoda Tabu_Search_Algorithm – metoda wykonująca kroki algorytmu zgodnie z pseudokodem z podpunktu 2.1.

2.2.7. Metoda generująca listę sąsiadów (GenerateNeighbourhood)- Nadrzędna metoda generująca sąsiedztwo o danym rozmiarze dla wybranego rozwiązania. Istotną cechą tej metody jest losowy wybór jednej z czterech funkcji tworzących pojedynczego sąsiada. Metoda ta upewnia się, że każde wygenerowane sąsiedztwo jest poprawne względem list tabu, oraz założeń problemu (wystarczająca ilość energii itp.)

2.2.8. Funkcje tworzące sąsiadów rozwiązania:
GenerateNeighbourSolution1 – funkcja wybierająca z rozwiązania punkt początkowy, końcowy oraz kilka/kilkanaście wybranych punktów pośrednich (bazowo 10) i na och podstawie oraz przy wykorzystaniu funkcji Find_Random_Solution tworzone jest nowe rozwiązanie.

GenerateNeighbourSolution2 – funkcja rozcinająca podane rozwiązanie w okolicach najgorszego punktu trasy – z największym kosztem czasu, uzyskanym za pomocą metody (WorstPoint). Następnie w tej okolicy losowany jest punkt pośredni. Tworzony jest nowy odcinek trasy rozpoczynający i kończący się między punktami krańcowymi rozciętego podanego rozwiązania oraz obejmujący punkt pośredni.
GenerateNeighbourSolution3 – funkcja rozcinająca podane rozwiązanie w okolicach najgorszego punktu trasy (punkty krańcowe rozciętej trasy to A i B). W porównaniu z poprzednią funkcją nie jest losowany punkt pośredni, lecz trasa między rozciętymi punktami prowadzona jest w liniach prostych – pionowej (lub poziomej) aż do osiągniecia danej współrzędnej takiej samej jak dla punktu B, a następnie poziomej (lub pionowej)) aż do osiągnięcia drugiej współrzędnej punktu B. W efekcie proces ten przypomina tworzenie narożnika.
GenerateNeighbourSolution4 – funkcja rozcinająca podane rozwiązanie w okolicach najgorszego punktu trasy (punkty krańcowe rozciętej trasy to A i B). Między punktami jest tworzona nowa trasa poprzez naprzemienne zmienianie współrzędnej, po której się poruszamy, aż do osiągniecia współrzędnych punktu B.

W zależności od konfiguracji, sposób dla danego sąsiada jest losowany listy wybranych sposobów.
Dla metod 2, 3 i 4 Co pewien czas (zgodnie z tym, jak zostało ustawione w RnP) zamiast najgorszego punktu brany jest losowy punkt z rozwiązania i wokół niego tworzone jest sąsiedztwo.

3. Aplikacja

3.1 Budowa aplikacji

Nasze stanowisko badawcze składa z głównych trzech kart:

- Mapy – umożliwia wyświetlenie mapy trudności terenu oraz mapy wysokości terenu. Mają postać HeatMap – im większa wartość poszczególnego piksela tym bardziej jego barwa zmierza ku czerwieni, a im mniejsza, tym bardziej niebieska;
- Rozwiązania – umożliwia wyświetlenie rozwiązania początkowego i najlepszego (z najmniejszym wymaganym czasem na przejazd). Rozwiązanie ma postać mapy binarnej – cała mapa jest w jednym kolorze, a trasa w drugim. Dodatkowo możliwe jest wyświetlenie trasy na tle jednej z map po kliknięciu odpowiedniego przycisku.
- Funkcja Celu – umożliwia wyświetlenie zmian najlepszego i obecnego rozwiązania w zależności od iteracji;

Dodatkowo, niezależnie od wybranej z kart z prawej strony aplikacji zawsze znajduje się panel ustawień – umożliwia:

- wgrywanie, zapisywanie i generowanie nowych map terenu;
- ustawianie pozycji punktu początkowego i końcowego;
- ustawianie parametrów łazika
- ustawianie parametrów algorytmu
- wybór sposobów generowania sąsiedztwa oraz jego rozmiaru;
- ustawienie liczby iteracji oraz uruchomienie algorytmu.
- przyciski map testowych

Do budowy stanowiska wykorzystaliśmy bibliotekę *PyQt5*, a wykresy i mapy generowane są z wykorzystaniem biblioteki *matplotlib*.

3.2. Format danych

W zagadnieniu mamy takie dane dotyczące badanego problemu jak:

- Mapa wysokości terenu – ma ona postać macierzy określającą wartość wysokości w poszczególnych punktach dyskretnych. Tworzona mapa przez generator jest zawsze skalowania między wartości 0 a 200, aby móc oszacować wstępnie wymagane zakresy nastaw pozostałych parametrów. Możliwe jest zapisanie wygenerowanej mapy jak i również wgranie własnej – wymagane jest wcześniejsze nazwanie pliku „HMatrix.csv".
  Wartości poszczególnych punktów są tym bardziej czerwone (brązowe) im mają większą wartość, a tym bardziej niebieskie im mniejszą.
- Mapa jakości terenu – ma ona postać macierzy określającą wartość jakości terenu (trudności w pokonaniu) w poszczególnych punktach dyskretnych. Im większa wartość w danym punkcie, tym więcej energii i czasu jest potrzebne na przejazd przez niego. Możliwe jest zapisanie wygenerowanej mapy jak i również wgranie własnej – wymagane jest wcześniejsze nazwanie pliku „TMatrix.csv".
  Wartości poszczególnych punktów są tym bardziej czerwone (brązowe) im mają większą wartość, a tym bardziej niebieskie im mniejszą.

- Pozostałe dane wprowadza się bezpośrednio w aplikacji w prawym panelu:
  - rozmiar map – edytowany w przypadku chęci wygenerowania mapy o zadanym rozmiarze przed uruchomieniem algorytmu.
  - współrzędne punktu początkowego i końcowego
  - parametry wyliczania czasu i energii: HCT, TCT, HCE, TCE (int) – domyślnie ustawiamy wszystkie na 1
  - parametry ładowania Sn1, Sn2 (int) – domyślne wartości to 8 i 4
  - energia początkowa łazika (int) – domyślnie ustawiona na 5000.

Należy również jeszcze sprecyzować dane algorytmu:

- KA+ – umożliwia aktywowanie i ustawienie co jaką liczbę iteracji ma wykonywać się drugie kryterium aspiracji w przypadku braku poprawy rozwiązania, domyślnie ustawione na 10.
- RnP – ustawienie co jaką liczbę iteracji zamiast najgorszego punktu ma być brany losowy w celu stworzenia nowego sąsiedztwa, domyślnie ustawione na 5.
- kT1, kT2 – są to kadencje istnienia rozwiązań (dla pierwszej) i punktów (dla drugiej) na listach tabu, domyślnie ustawione na 30 i 10.
- sposoby generowania sąsiedztwa oraz jego rozmiaru (Size)
- liczba iteracji

3.3. Wymagania odnośnie uruchomienia

W celu uruchomienia projektu wymagany jest zainstalowany Python 3.8 oraz biblioteki:

- matplotlib – biblioteka umożliwiająca tworzenie map i wykresów
- numpy – biblioteka ułatwiająca pracę z macierzami
- PyQt5 – biblioteka umożliwiająca stworzenie aplikacji (GUI)

Główny plik algorytmu to **main.py**, natomiast plik uruchamiający aplikację to **gui_demo.py.** Dwa pliki (**generation_methods.py**, **Generate_Map.py**) są wykorzystywane przez plik main.py, a plik **App.py** przez gui_demo.py.

Po uruchomieniu pliku z aplikacji mamy na początku 3 możliwości:

- Wgrania jednego z 3 zestawów map testowych – służą do tego przyciski **Test1, Test2, Test3**.
- Wgrania własnego zestawu map, który został wcześniej zapisany lub wygenerowany w innym pliku/miejscu, za pomocą przycisku **Wczytaj mapy**
- Wygenerowania nowego zestawu map – w tym celu należy ustawić żądaną wartość rozmiaru mapy, a następnie kliknąć przycisk **Wygeneruj mapy**.

Po wygenerowaniu map możliwe jest ich również zapisanie za pomocą przycisku **Zapisz mapy**.

Następnie ustawiamy parametry problemu i algorytmu – każdy z nich został opisany w punkcie 3.3. Po ustawieniu parametrów klikamy **Uruchom algorytm** i czekamy na rezultaty.

3.4. Format wyników

Wynikami działania aplikacji są takie dane jak:

- Rozwiązanie początkowe i najlepsze – bazowo mają one postać list nazwanych krotek ze współrzędnymi poszczególnych punktów trasy łazika. Jednak w celu wizualizacji punkty te naniesiono na zerową macierz (uprzednio przygotowaną o rozmiarze map) i nadano im takie same wartości. Dzięki temu możliwe jest ich wyświetlenie tak jak map terenu w postaci binarnej (trasa łazika w jednym kolorze, tło w drugim). Dodatkowo możliwe jest wybranie opcji wyświetlenia tras na tle mapy wysokości terenu za pomocą przycisku **H**, na tle mapy jakości terenu za pomocą przycisku **T** oraz powrót z tego trybu za pomocą przycisku **N**.

- Tablice wartości czasu dla rozwiązania najlepszego i aktualnego dla poszczególnych iteracji. Na ich podstawie tworzone są wykresy prezentujące powyższe dane. Przedstawiają zachowanie się algorytmu w różnych przedziałach czasu jego działania.

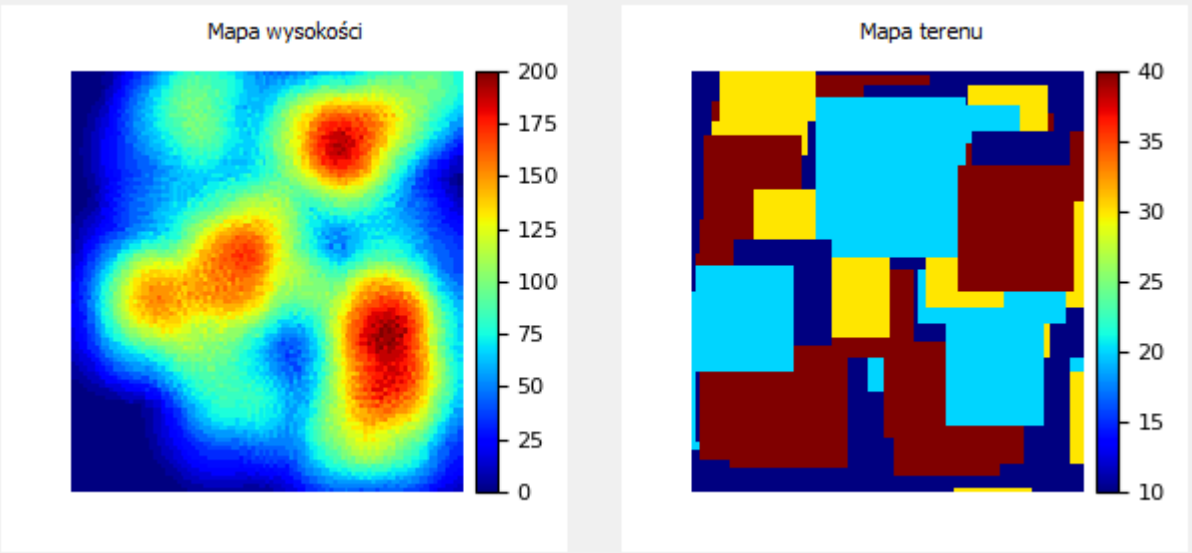
4. Testy

Test 1 Porównanie działania algorytmu dla różnych konfiguracji sąsiedztw

Autor: Marcin Brzózka

Celem testu było porównanie uzyskanych najlepszych czasów w zależności od wyboru metod generujących sąsiedztwo. Dla każdej konfiguracji wykonano po 10 wywołań algorytmu w celu uśrednienia uzyskanych wyników.

Test był przeprowadzany na 1 testowym zestawie map (o rozmiarze 100x100).



Rys. 1 Wykorzystane mapy w teście

Ustawione parametry wynosiły:

- Start: Point(5, 5), End: Point(80, 80)
- HCE = 1, TCE = 1, HCT = 3, TCT = 1, Sn1 = 8, Sn2 = 4, Energia = 3000
- kA+ = 10, RnP = 4, rozmiar sąsiedztwa = 20, kT1 = 30, kT2 = 10
- Liczba iteracji = 1000, dodatkowy war. zakończenia - wyłączony

Średnia wartość czasu pierwszego rozwiązania wynosiła około 4590.

Wyniki prezentują się następująco:

Wybrane sposoby	Średnia wartość czasu dla najlepszego rozwiązania
1	3868.320
2	3717.160
3	4334.400
4	4561.080
1 i 2	3478.120
1 i 3	3764.160

1 i 4	3758.960
2 i 3	3645.160
2 i 4	3840.760
3 i 4	4413.800
1, 2 i 3	3522.820
1, 2 i 4	3628.660
1, 3 i 4	3741.160
2, 3 i 4	3721.120
wszystkie	3595.740

Analizując powyższe wyniki można zauważyć, że najlepsze rezultaty uzyskujemy dla wyboru 1 i 2 sposobu generowania sąsiedztwa, natomiast najgorsze (bliskie rozwiązaniu startowemu) dla 4 sposobu.
Można zauważyć, że dla tych konfiguracji, gdzie występują 1 lub / oraz 2 sposób generowania sąsiedztwa uzyskane czasy są znacznie lepsze niż dla pozostałych.
Tak uzyskane wyniki można było przewidzieć, ponieważ to właśnie te dwie pierwsze metody umożliwiają największe zmiany przy tworzeniu sąsiedztwa. Dzięki temu jest możliwe porównanie znacznie większej liczby kombinacji i tym samym znalezienie jak najlepszego rozwiązania. Umożliwiają znalezienie go również w znacznie mniejszej liczbie iteracji niż gdybyśmy chcieli uzyskać podobne czasy dla pozostałych dwóch metod.

Test 2 Porównanie działania algorytmu dla różnych kadencji obu list tabu

Autor: Marcin Brzózka

Celem testu było porównanie uzyskanych najlepszych czasów w zależności od nastaw kadencji obu list tabu. Dla każdej konfiguracji wykonano po 10 wywołań algorytmu w celu uśrednienia uzyskanych wyników.
Test był przeprowadzany na 1 testowym zestawie map (o rozmiarze 100x100).
Ustawione parametry wynosiły:

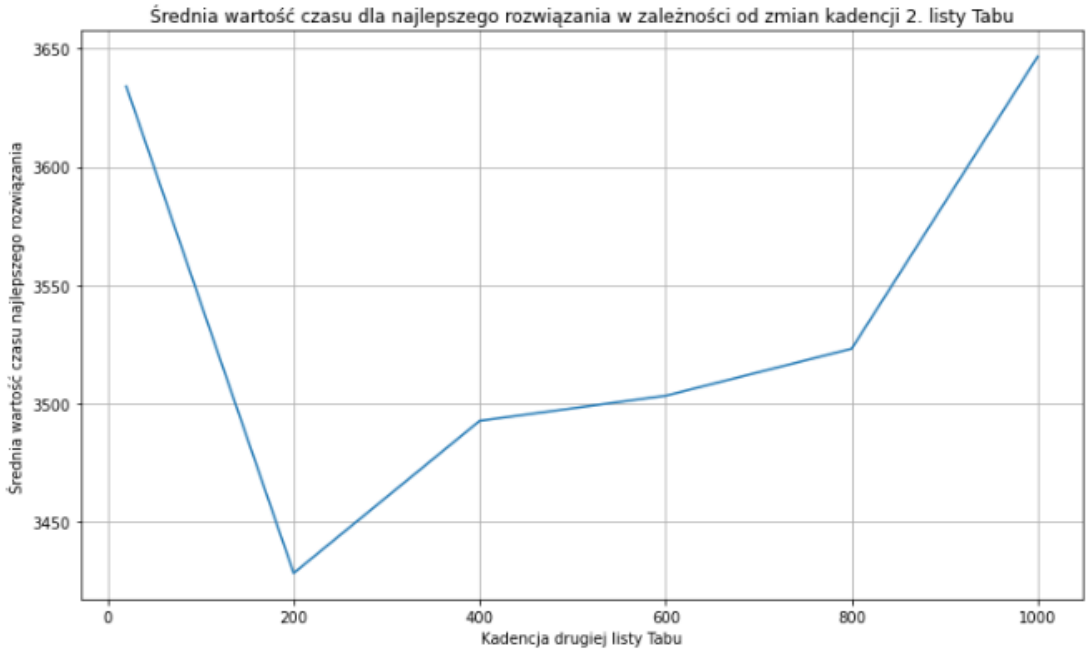
- Start: Point(5, 5), End: Point(80, 80)
- HCE = 1, TCE = 1, HCT = 3, TCT = 1, Sn1 = 8, Sn2 = 4, Energia = 3000
- kA+ = 10, RnP = 4, rozmiar sąsiedztwa = 20, wszystkie sposoby generowani sąsiedztwa
- Liczba iteracji = 1000, dodatkowy war. zakończenia - wyłączony

Zmiany kadencji pierwszej listy Tabu (kadencja drugiej listy stała = 200)

Kadencja 1. listy Tabu	**Średnia wartość czasu dla najlepszego rozwiązania**
20	3624.580
200	3419.080
400	3304.180
600	3409.900
800	3616.880
1000	3623.460

Zmiany kadencji drugiej listy Tabu (kadencja pierwszej listy stała = 200)

Kadencja 2. listy Tabu	**Średnia wartość czasu dla najlepszego rozwiązania**
20	3634.100
200	3428.220
400	3492.620
600	3503.240
800	3523.140
1000	3646.720



Możemy zauważyć, że zmiany kadencji dla poszczególnych list Tabu mają wpływ na osiągnięcie najlepszego rozwiązania, lecz nie jest on znaczeni różniący się dla poszczególnych przypadków. Dla pierwszej listy najlepsze rezultaty otrzymujemy gdy kadencja jest ustawiona na 400, a dla drugiej gdy jest ustawiona na 200. Taka różnica wynika z faktu, iż o wiele trudniej algorytmowi trafić w dokładnie takie samo rozwiązanie niż w zestaw punktów. Zmiany wartości kadencji nie wpływają w widoczny sposób na szybkość działania algorytmu.

Należy jednak dodać, że takie przebiegu uzyskano dopiero za drugim / trzecim uruchomieniem testu. Ze względu na dużą losowość nie zawsze uzyskujemy takie wyniki, jakich byśmy oczekiwali. Czasem zmiany są niewielkie, a czasem powstawały ostre piki. Należy jednak zaznaczyć, że dla najkrótszych i najdłuższych list tabu czasy wychodziły najdłuższe. Podsumowując można zauważyć wpływ wartości kadencji list tabu na rozwiązanie, ale wpływ ten nie jest duży.

Test 3 Porównanie działania algorytmu dla różnych rozmiarów sąsiedztwa

Autor: Marcin Brzózka

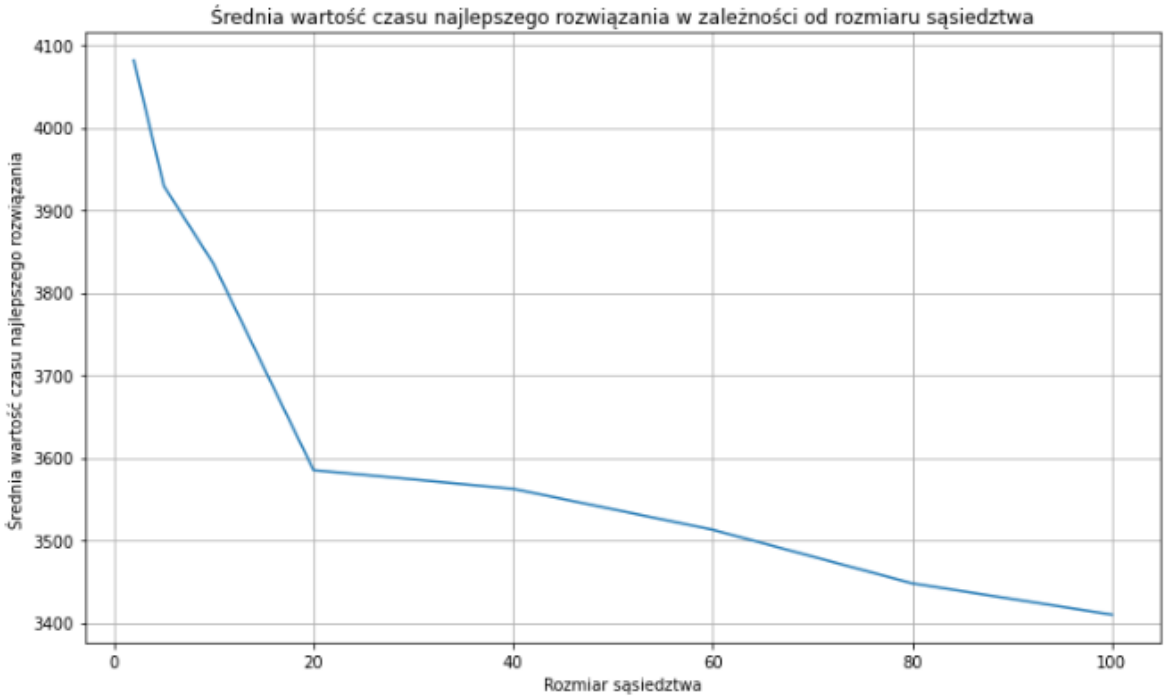
Celem testu było porównanie uzyskanych najlepszych czasów w zależności od nastawy rozmiaru sąsiedztwa. Dla każdej konfiguracji wykonano po 10 wywołań algorytmu w celu uśrednienia uzyskanych wyników.
Test był przeprowadzany na 1 testowym zestawie map (o rozmiarze 100x100).

Ustawione parametry wynosiły:

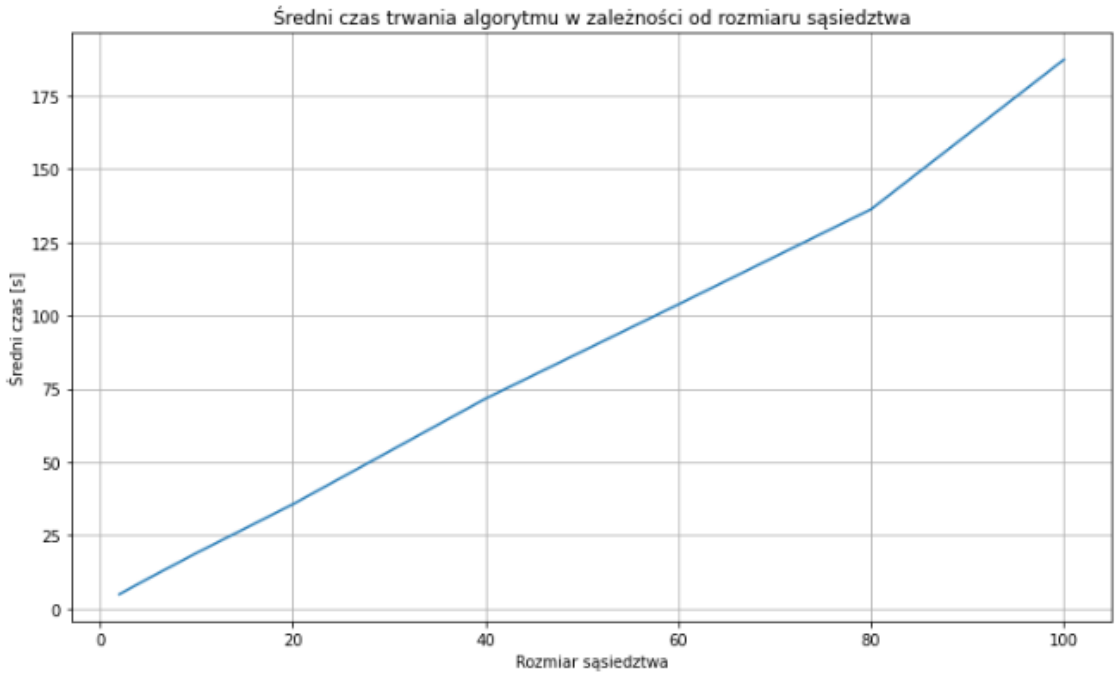
- Start: Point(5, 5), End: Point(80, 80)
- HCE = 1, TCE = 1, HCT = 3, TCT = 1, Sn1 = 8, Sn2 = 4, Energia = 3000
- kA+ = 200, RnP = 4, wszystkie sposoby generowani sąsiedztwa, kT1 = 200, kT2 = 200
- Liczba iteracji = 1000, dodatkowy war. zakończenia – wyłączony

Średnia wartość czasu pierwszego rozwiązania wynosiła około 4482.
Rezultaty testu:

Rozmiar sąsiedztwa	Średnia wartość czasu dla najlepszego rozwiązania
2	4081.400
5	3929.280
10	3834.380
20	3584.880
40	3562.220
60	3512.620
80	3447.680
100	3409.820



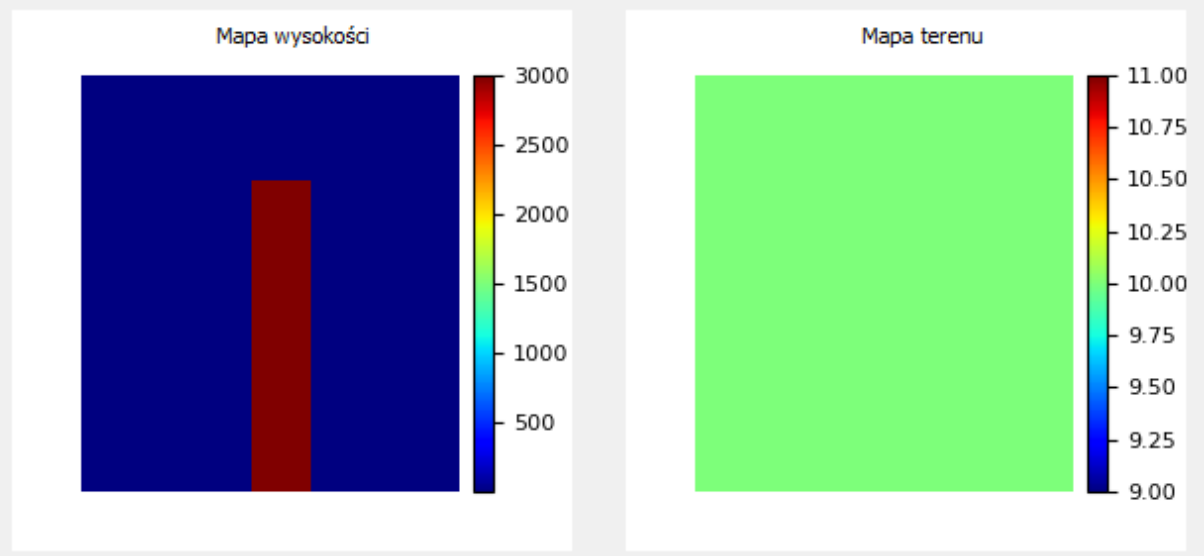
Na podstawie powyższej tabeli można zauważyć, że rozmiar sąsiedztwa ma znaczenie. Największe różnice wpływu rozmiaru sąsiedztwa można zauważyć dla jego zmian do rozmiaru = 20 – im większy rozmiar, tym większa ilość możliwych rozwiązań dla poszczególnych iteracji i tym samym większa szansa na znalezienie lepszego rozwiązania. Dla dalszego zwiększania rozmiaru dalej mogliśmy obserwować poprawy, lecz nie były już tak znaczne jak wcześniej. Wynika to z faktu, że przy rozmiarze 20 mieliśmy już wystarczająco wiele próbek do porównania i byliśmy bardzo blisko najlepszego możliwego rozwiązania dla mapy o rozmiarze 100x100. Aby polepszyć wynik warto by było tutaj popróbować zmieniać inne parametry, ponieważ zwiększanie rozmiaru sąsiedztwa znacznie wydłuża wykonywanie się algorytmu.
Dla większych map rozmiar sąsiedztwa umożliwiający stosunkowo szybkie znalezienie dobrego rozwiązania może być większy.



Test 4 Testowanie omijania przeszkody

Autor: Marcin Brzózka

Celem tego testu było przetestowanie algorytmu w pokonywaniu przeszkody. W tym celu został przygotowany zestaw map **Test 2**. Energia została dobrana tak, aby uniemożliwić próbę wjechania na przeszkodę.

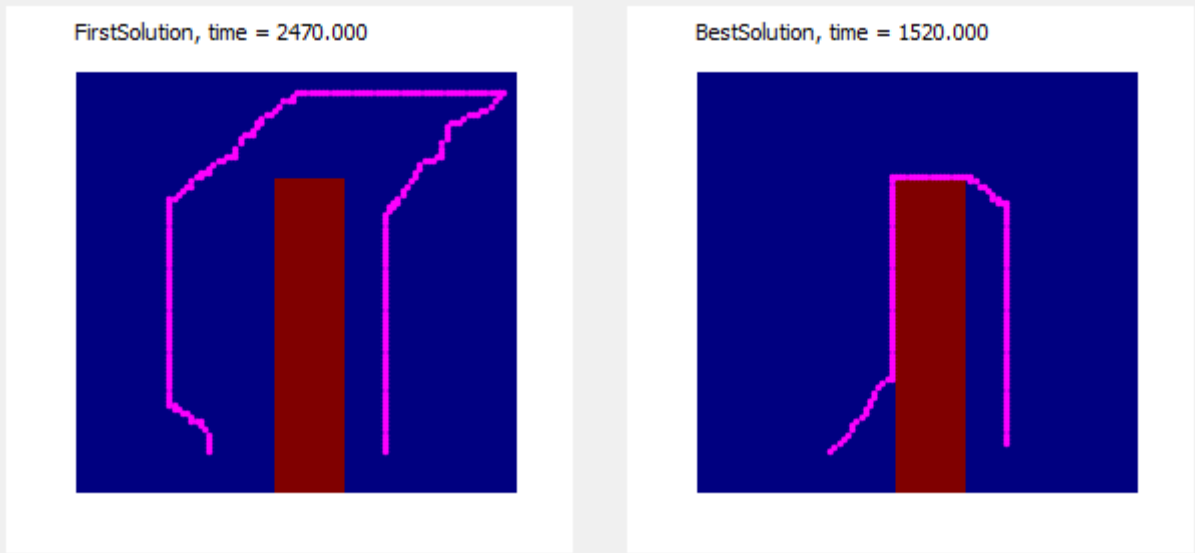


Rys. 2 Wykorzystane mapy w teście

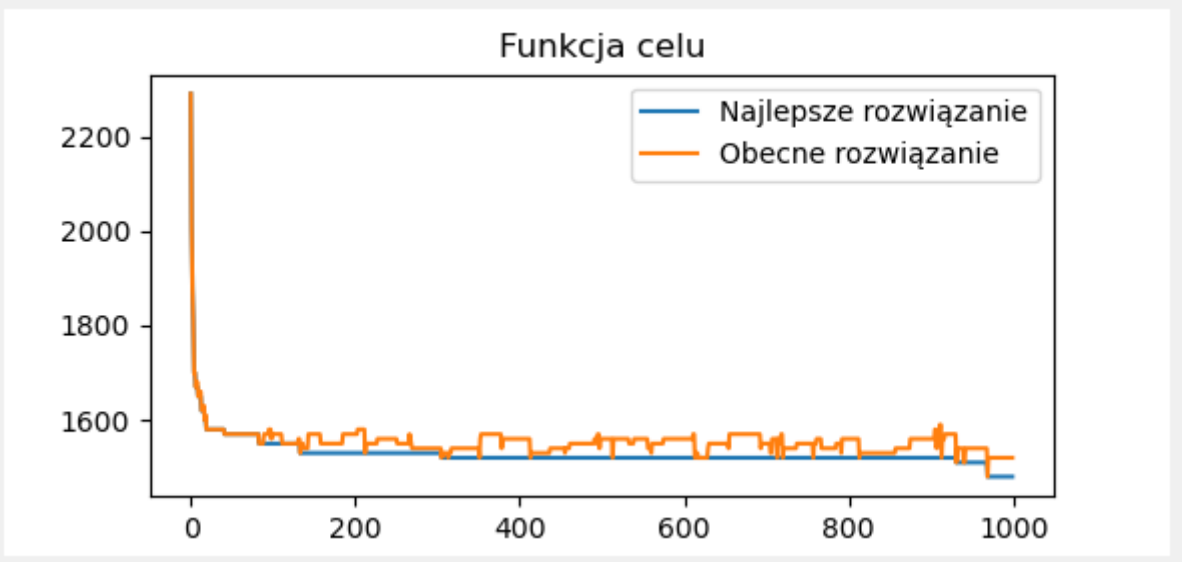
Ustawione parametry wynosiły:

- Start: Point(30, 10), End: Point(70, 10)
- HCE = 1, TCE = 1, HCT = 3, TCT = 1, Sn1 = 8, Sn2 = 4, Energia = 3000
- kA+ = 100, RnP = 4, rozmiar sąsiedztwa = 20, wszystkie sposoby generowani sąsiedztwa, kT1 = 200, kT2 = 200
- Liczba iteracji = 1000, dodatkowy war. zakończenia - wyłączony

Rezultaty wykonania się algorytmu:



Rys. 3 Trasy łazika – początkowa i dla najlepszego rozwiązania na tle przeszkody na mapie wysokości terenu



Rys. 4 Przebiegi funkcji obecnego i najlepszego rozwiązania

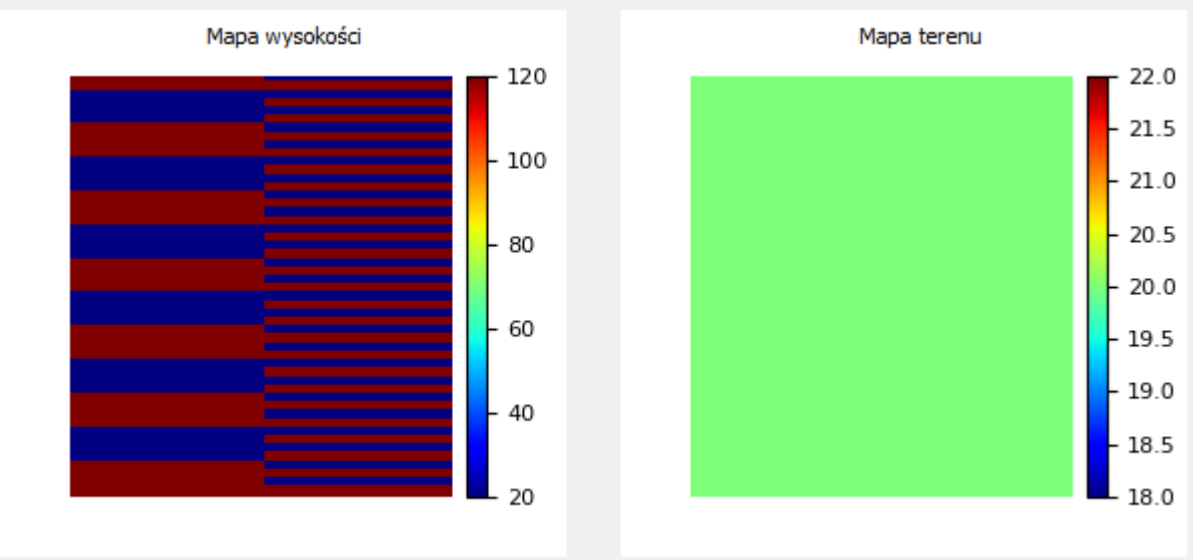
Można zauważyć, że algorytm poradził sobie z problemem przeszkody. Było to możliwe głównie dzięki znalezieniu pierwszego rozwiązania, które omijało przeszkodę. Dodatkowo w kolejnych iteracjach trasa została tak poprawiona, że łazik przejeżdża blisko przeszkody na jej końcu minimalizując tym samym czas.

Taka przeszkoda z łatwością jest omijana. Problem może nastąpić w przypadku bardziej skomplikowanych przeszkód – wtedy należy zmodyfikować funkcję tworzącą pierwsze rozwiązanie.

Test 5 Specyficzny kształt mapy

Autor: Marcin Brzózka

Celem tego testu było przetestowanie algorytmu w pokonywaniu specyficznego zestawu map, który został zapisany pod **Test 3**.



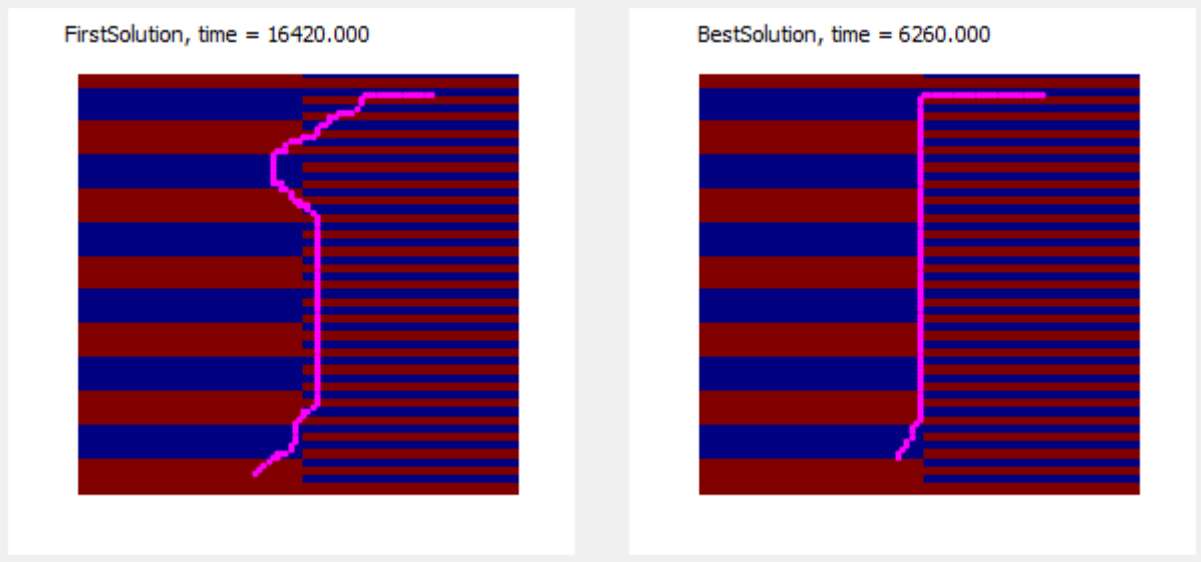
Rys. 5 Wykorzystane mapy w teście

Mapa jakości terenu jest stała, dlatego nie ma dużego wpływu na przebieg trasy. Natomiast mapa wysokości terenu podzielona jest pionowo na dwa sektory, w których wysokość zmienia z 20 na 120 i z 120 na 20 co pewną ilość punktów w osi pionowej - dla lewego sektora co 8, a dla prawego co 2.

Test ma za zadanie wychwycenie, czy większa cześć trasy będzie przebiegać przez lewy sektor, w których rzadziej dochodzi do zmian wysokości i tym samym koszt przejazdu jest mniejszy.

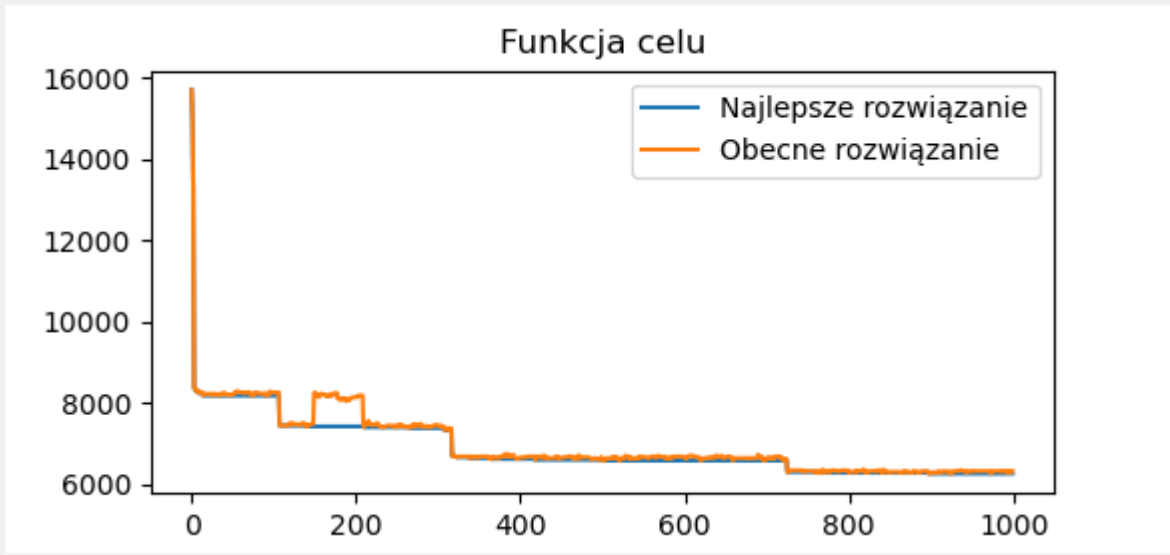
Ustawione parametry wynosiły:

- Start: Point(40, 5), End: Point(80, 95)
- HCE = 1, TCE = 1, HCT = 5, TCT = 1, Sn1 = 0, Sn2 = 0, Energia = 3000
- kA+ = 100, RnP = 5, rozmiar sąsiedztwa = 20, wszystkie sposoby generowani sąsiedztwa, kT1 = 200, kT2 = 200
- Liczba iteracji = 1000, dodatkowy war. zakończenia – wyłączony



Rys. 6 Trasy łazika – początkowa i dla najlepszego rozwiązania na mapie wysokości terenu

Możemy zauważyć, że algorytm zachował się zgodnie z założeniami. Po podjechaniu do miejsca zmiany sektorów, pierwszy sektor nie jest zmieniany na drugi, aż łazik podjedzie do punktu, z którego w linii prostej po osi X może przemieścić się do celu. Dzięki temu zaoszczędzony jest czas, którego więcej by minęło przy przemieszczaniu się w osi pionowej przez sektor drugi.



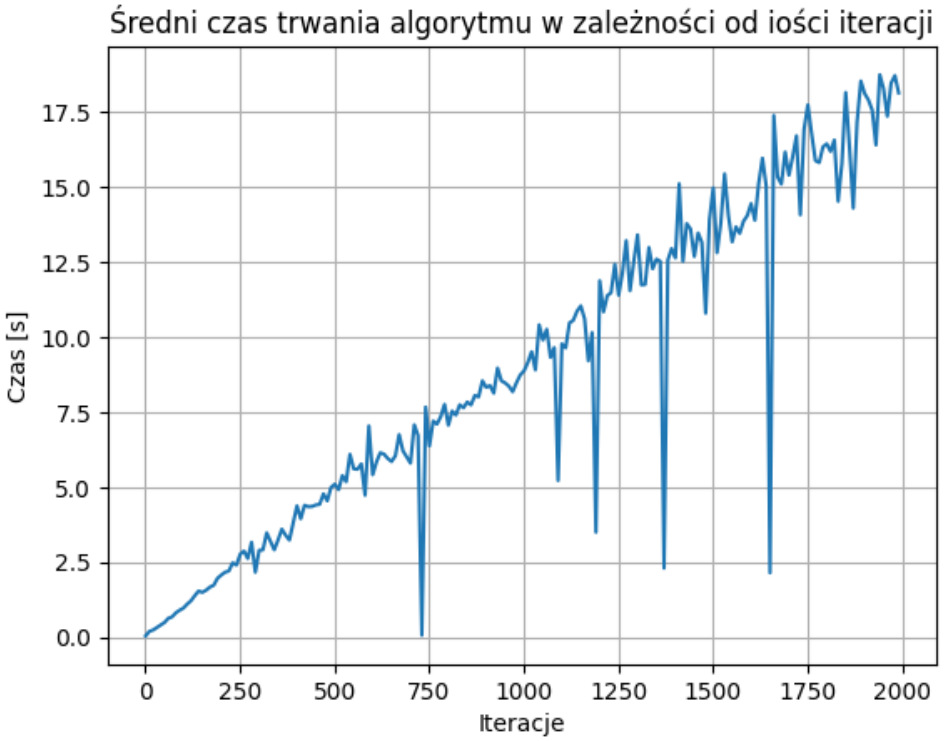
Rys. 9 Przebiegi funkcji obecnego i najlepszego rozwiązania

Możemy tutaj też zauważyć jak stopniowo najlepsze rozwiązanie się poprawiało wraz ze zjazdem z prawego sektora.

Test 6 Pomiar czasu wykonywania algorytmu w zależności od ilości iteracji

Autor: Stanisław Dudiak

W celu dokonania powyższego pomiaru została przeprowadzona seria testów. Czas wykonywania algorytmu został zmierzony i uśredniony dla różnych ilości iteracji. Wyniki pomiarów zostały pokazane na poniższym wykresie.



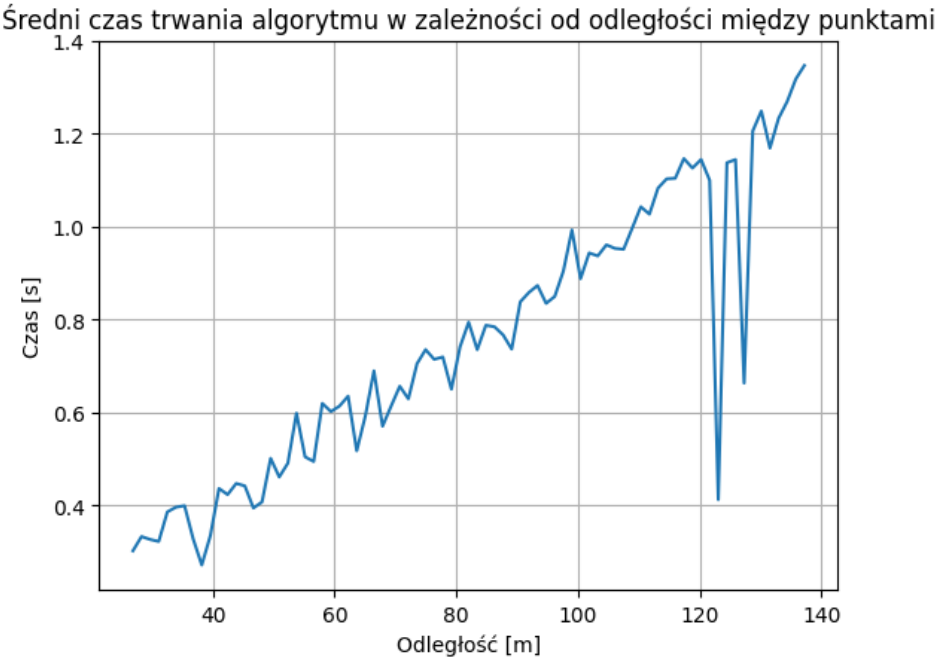
Rys. 10 Wykres czasu wykonywania algorytmu w zależności od liczby iteracji

Można zauważyć, że czas wykonywania algorytmu rośnie liniowo wraz ze wzrostem ilości iteracji. Gwałtowne skoki w dół na wykresie są spowodowane faktem, że przez określoną ilość iteracji nie zostało znalezione lepsze rozwiązanie i algorytm zakończył działanie.

Test 7 Pomiar czasu wykonywania algorytmu w zależności od dystansu między punktem końcowym a początkowym

Autor: Stanisław Dudiak

Testy zostały wykonane w identyczny sposób jak poprzednio, jednak zmienną nie był czas trwania algorytmu, a odległość między punktem końcowym i początkowym.



Rys. 11 Wykres czasu wykonywania algorytmu w zależności od odległości między punktami początkowym i końcowym

5. Podsumowanie

5.1. Wnioski

Stworzony algorytm i aplikacja dosyć dobrze radzą sobie z problemem utworzenia o minimalnym czasie dla łazika marsjańskiego. Można to zauważyć analizując wyniki przeprowadzonych testów. Testy dla drugiego i trzeciego przygotowanego zestawu map wskazują, że trasa jest tak tworzona, by minimalizować czas i omijać przeszkody. Test rozmiaru sąsiedztwa udowodnił, że ma on znaczenie i że czym większy tym większa szansa na znalezienie lepszego rozwiązania końcowego w ograniczonej ilości iteracji oraz że zależność ta nie jest liniowa (przy pewnym rozmiarze różnice te zaczynają zanikać). Test wartości kadencji wskazał, że są pewne wartości, przy których możliwe jest znalezienie najlepszego rozwiązania przy pewnej liczbie iteracji oraz że dla wartości większych zabraniamy za mało obecnych rozwiązań a dla większych za dużo i tym samym blokujemy najkrótszą ścieżkę do rozwiązania optymalnego.

5.2. Stwierdzone problemy

Jednym z napotkanych problemów była minimalna pewna minimalna odległość między punktem startowym a końcowym. Gdy zmniejszaliśmy odległość poniżej 20 punktów, niektóre funkcje losujące nie mogły działać poprawnie. Rozwiązaniem problemu jest uzależnienie ilości losowanych punktów od długości trasy łazika, a w przypadku punktów obok siebie (na ukos) przetestowanie jedynie trzech możliwych sposobów dotarcia – pionowo, poziomo i na ukos.

Wartość kadencji ma znaczenie, lecz nie ma ono zbyt dużego wpływu.

5.3. Kierunki dalszego rozwoju

W przyszłości można by się zastanowić na dodanie opcji restartu algorytmu w przypadku nie znalezienia lepszego rozwiązania przez dłuższy czas (czyli np. w sytuacji zadziałania drugiego kryterium zakończenia). Taka opcja mogłaby pomóc w niektórych sytuacjach, gdy pierwsze rozwiązanie i najlepsze krąży wokół jednego minimum czasu i w żaden sposób nie potrafi się przedostać do drugiego, które jest minimum globalnym. W takiej sytuacji jest wymagane spróbowanie znalezienie pierwszego rozwiązania z dala od obecnego minimum i tym samym bliżej drugiego.

Można się również zastanowić nad poprawieniem funkcji znajdującej pierwsze rozwiązanie, a dokładniej na stworzeniu funkcji badającej obszary mapy i tym samym stworzenia pierwszej trasy przez mniej wymagające punkty. Dzięki temu możliwe by było pokonywanie bardziej skomplikowanych tras z przeszkodami – labiryntu.

Należałoby się również się przyjrzeć jeszcze listom tabu i spróbować je dopracować, aby odpowiednie nastawy kadencji dla danego przypadku dawały zawsze najlepsze wyniki i tym samym zmniejszyć istniejącą losowość i ich niewielki wpływ.