Projekt Optymalizacja Wielokryterialna

Realizacja SWD implementującego metody optymalizacji wielokryterialnej

Autorzy:

Julia Nowak

Adam Złocki

Jakub Szczypek

Spis treści

Podział zadań

Członek zespołu	Zadania
Julia Nowak	• Przygotowanie GUI • Dokumentacja – wstęp i przygotowanie danych • Wstępna wersja kodu PSO
Adam Złocki	• Poprawki zaimplementowanego już algorytmu PSO • Implementacja drugiego algorytmu (jeszcze niezdefiniowany) • Praca nad dokumentacją
Jakub Szczypek	• Praca nad zaimplementowanym już algorytmem PSO – wprowadzanie poprawek i rozwijanie aktualnej wersji algorytmu • Praca nad dokumentacją

Wstęp

Problem badawczy

Tematem projektu jest **optymalizacja przejazdu zespołu robotów przez przeszkodę**, co stanowi przykład problemu wielokryterialnego w optymalizacji. Problem ten dotyczy znalezienia kompromisu między dwoma sprzecznymi kryteriami: minimalizacją uszkodzeń robotów a minimalizacją czasu potrzebnego na pokonanie przeszkody przez cały zespół.

Kluczowe założenia problemu to:

- **Zakłócenia w ruchu robotów** na przeszkodzie mogą prowadzić do ich uszkodzeń, a ryzyko to rośnie w przypadku wysokiej gęstości robotów w jednym obszarze.
- **Dystans między robotami** wpływa na poziom zakłóceń i ryzyko uszkodzeń – większy dystans zmniejsza ryzyko, ale zwiększa czas przejazdu.

Głównym celem badawczym jest zaprojektowanie i wdrożenie efektywnego algorytmu wielokryterialnej optymalizacji, który uwzględnia oba te kryteria.

Opis wykorzystywanej bazy danych / zbioru decyzyjnego

W projekcie do symulacji i analizy danych planujemy wykorzystać różne źródła danych:

1. **Generowane dane testowe**
   Przygotujemy sztuczne dane reprezentujące różne konfiguracje przeszkód i zakłóceń. Dane te będą obejmować:
   - Mapy 2D/3D z różnorodnymi układami przeszkód (np. linie proste, labirynty, przeszkody dynamiczne).
   - Zespoły robotów o różnej liczebności i właściwościach.

2. **Dane z literatury naukowej**
   Skorzystamy z prac dotyczących swarm robotics oraz optymalizacji trajektorii, które dostarczają przykładów i modeli problemów podobnych do analizowanego w naszym projekcie.

3. **Publiczne bazy symulacyjne**
   Wykorzystamy otwarte źródła, takie jak:
   - Open Robotics,
   - ROS (Robot Operating System),
   - Bazy danych i środowiska symulacyjne dostępne w repozytoriach naukowych, np. artykuły z *ScienceDirect*.

   Dzięki połączeniu danych generowanych, literaturowych oraz z publicznych baz symulacyjnych, zapewnimy szeroki kontekst i realistyczne warunki do optymalizacji oraz weryfikacji opracowanego algorytmu.

Przykładowymi źródłami danych mogą być:

- https://library.fiveable.me/swarm-intelligence-and-robotics
- https://en.wikipedia.org/wiki/Particle_swarm_optimization
- https://www.sciencedirect.com/science/article/pii/S0304389424004114

Przygotowanie i weryfikacja danych

Opis ogólny

Do przygotowania mapy terenu, która jest główną informacją wejściową naszego algorytmu, przygotowaliśmy funkcję generującą mapę wysokościową terenu. Proces generowania mapy terenu służy do symulacji różnych typów wysokości terenu w formie macierzy danych, które mogą być wizualizowane w postaci mapy wysokości. Funkcja umożliwia generowanie różnych struktur terenu, takich jak pagórki, linie, nachylenia, kaniony, wzory przypominające labirynty oraz inne nieregularne kształty.

Parametry

Funkcja posiada następujące argumenty wejściowe:

1. **noise_num** (liczba, domyślnie 1)

   Określa intensywność losowych zakłóceń (szumu) dodanych do mapy terenu. Większe wartości generują bardziej chaotyczne powierzchnie.

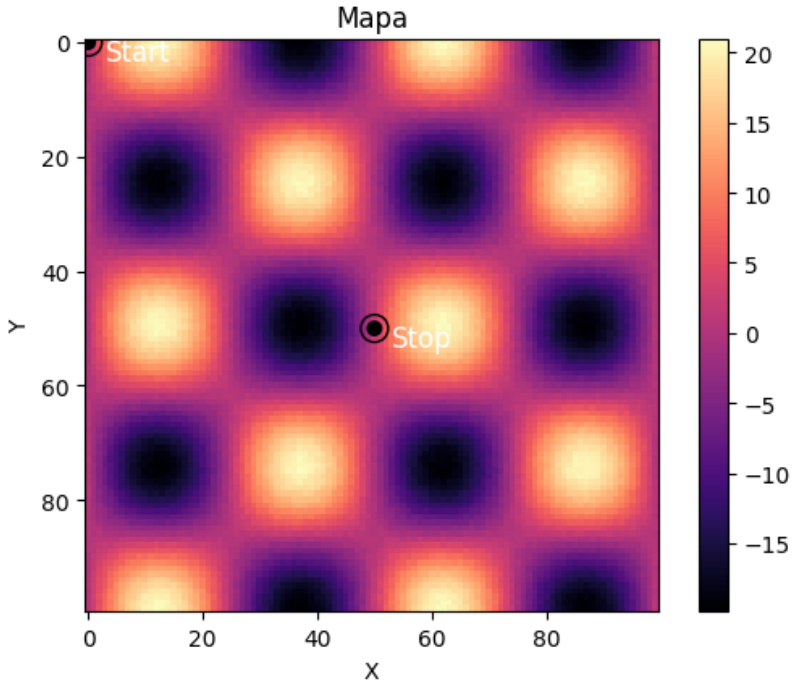
2. **terrain_size** (krotka, domyślnie (100, 100))

   Rozmiar generowanego terenu w postaci liczby punktów w osiach X i Y. Na przykład (100, 100) oznacza macierz 100x100 pikseli.

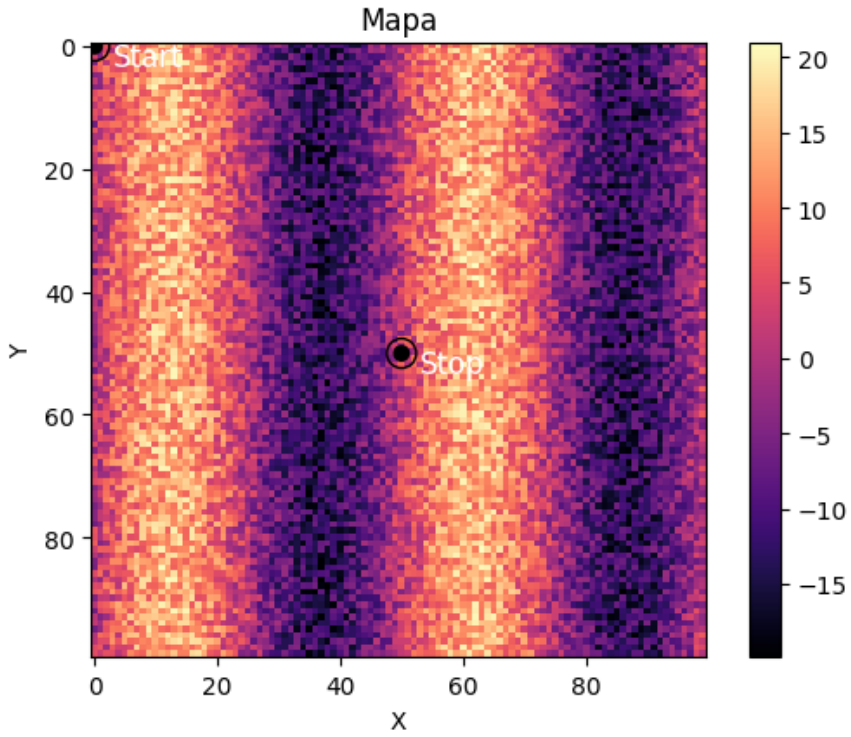
3. **terrain_type** (łańcuch znaków, domyślnie "hills")

   Typ struktury terenu, który ma zostać wygenerowany. Dostępne tryby:

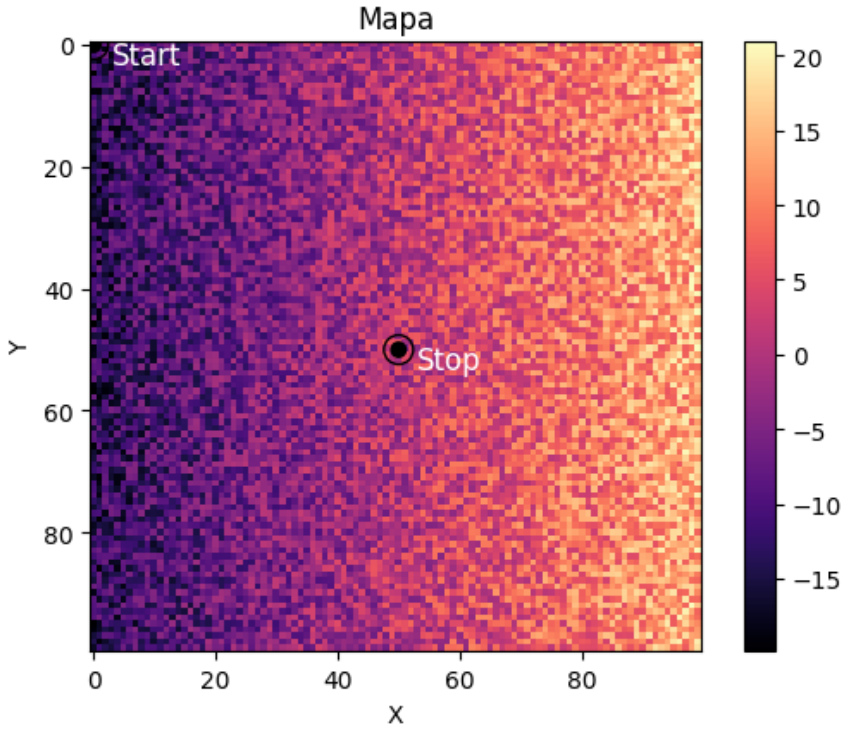
   - **"hills":** Pagórkowata powierzchnia oparta na sinusoidalnych wzorach.



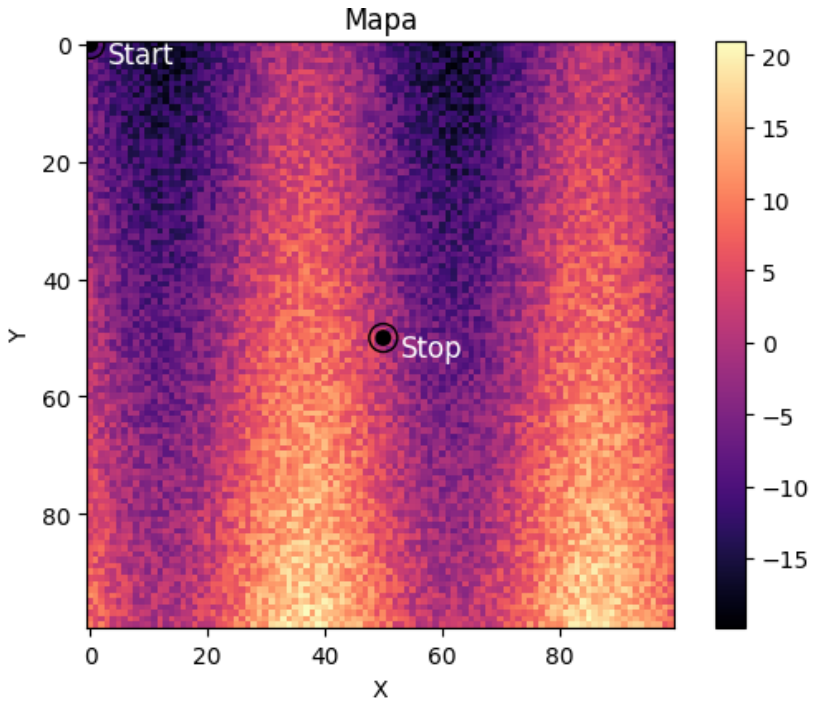
- **"lines":** Powierzchnia z liniowymi wzorami wzdłuż osi X.



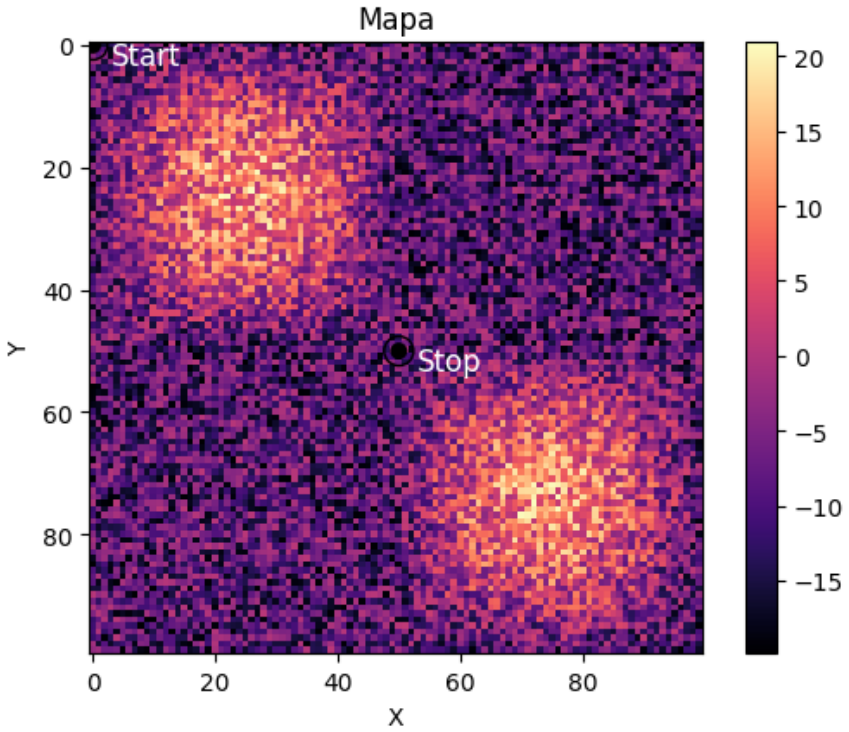
- **"slope":** Jednostajny spadek wzdłuż osi X.



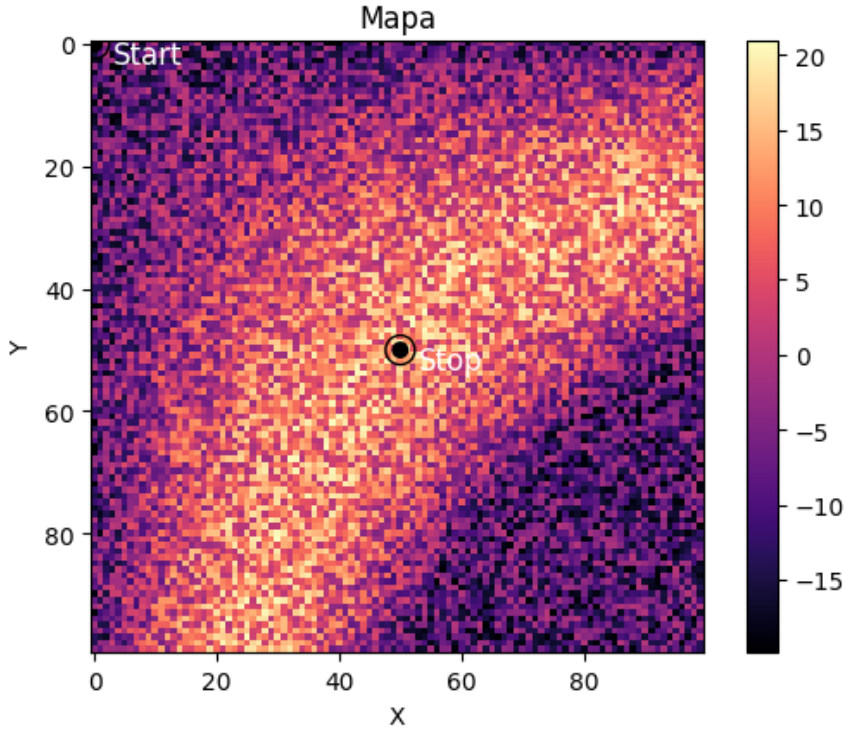
- **"razors":** Powierzchnia z ostrymi przejściami wzdłuż osi X i Y.



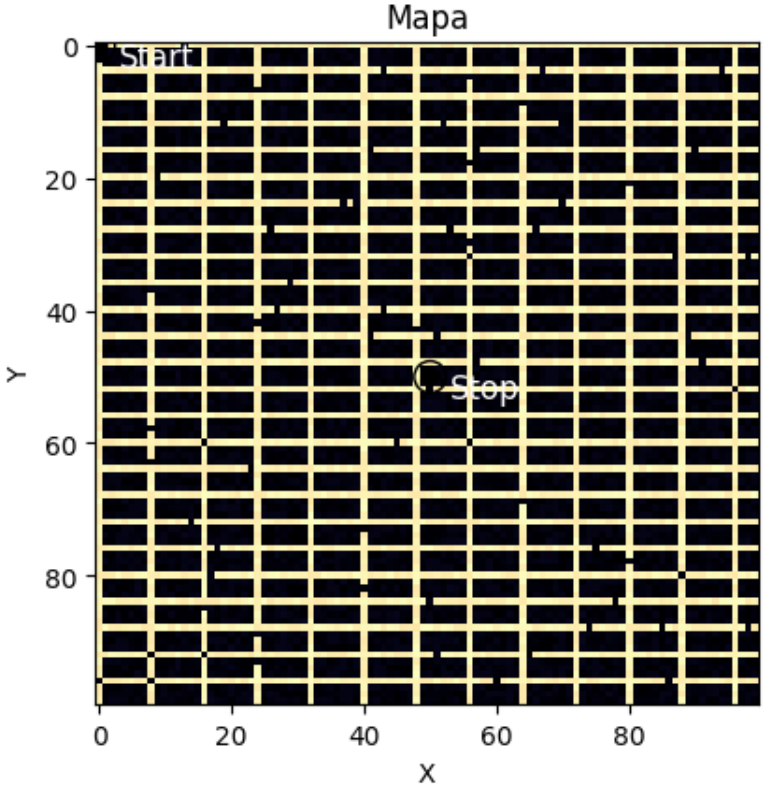
- **"canyon":** Powierzchnia przypominająca kanion z dodatkowymi dolinami.



- **"bow":** Powierzchnia w kształcie łuku oparta na iloczynie sinusoidalnym.



- **"maze":** Wzór przypominający labirynt, zawierający ściany oraz przypadkowe otwarcia w ścianach.



Działanie funkcji

Na początku generowane są losowe wartości w postaci macierzy o rozmiarze określonym przez terrain_size. Te wartości reprezentują losowy szum, który będzie bazą dla struktury terenu. Następnie generowana jest siatka współrzędnych dla osi X i Y, umożliwiająca modelowanie wzorów sinusoidalnych lub liniowych dla wybranej struktury terenu.

Na podstawie wartości terrain_type tworzony jest odpowiedni wzór, np. pagórki, linie lub nachylenia lub generowany jest labirynt. Wygenerowana struktura jest łączona z zakłóceniami w celu stworzenia ostatecznej mapy terenu.

Wybór rozmiaru macierzy (terrain_size) wpływa na poziom szczegółowości wygenerowanej mapy. Większe rozmiary mogą zwiększyć dokładność, ale wydłużają czas obliczeń. Intensywność zakłóceń (noise_num) może być regulowana w celu uzyskania bardziej realistycznych lub abstrakcyjnych wyników.

Implementacje metody wielokryterialnej

Jako metodą wielokryterialną proponujemy zastosowanie rankingowania z **RSM** i **VIKOR**.

Implementacje metody wielokryterialnej :

- wdrożenie metod optymalizacji parametrów / Implementacja dodatkowej metody jako metody odniesienia

Analiza wyników

- wyliczone metryki z komentarzami,

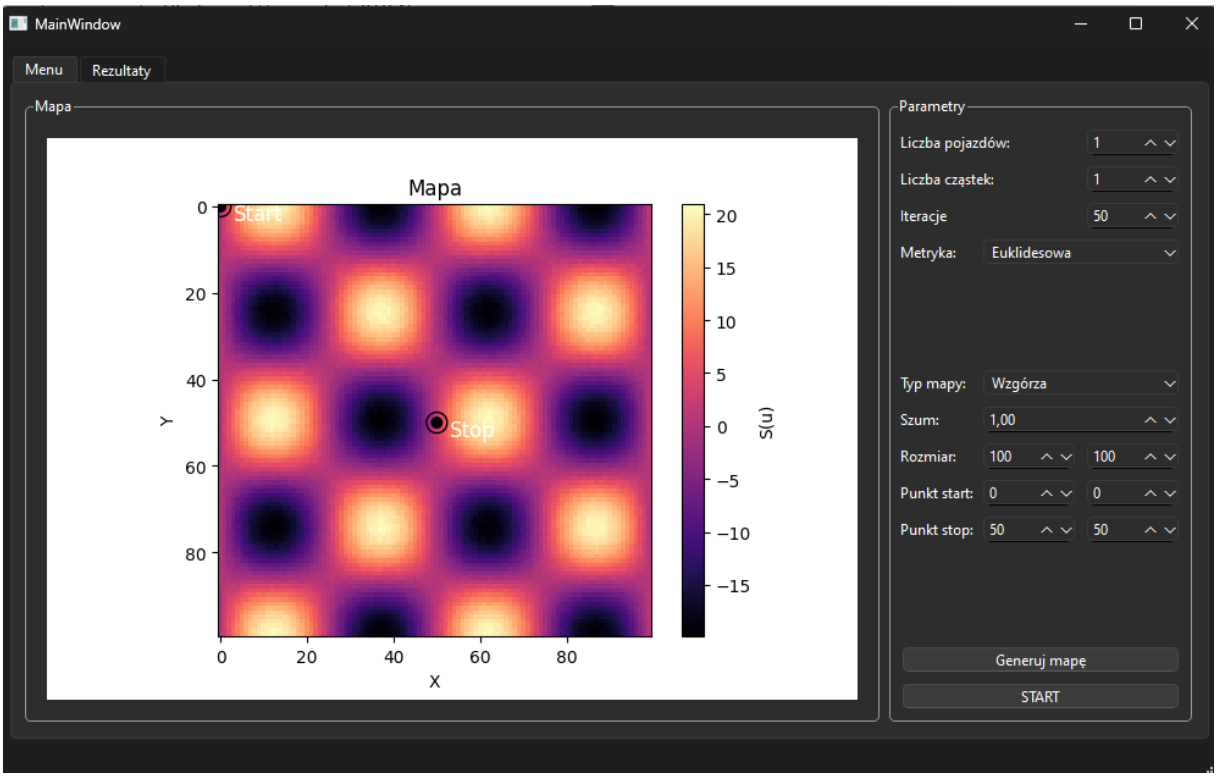
- optymalizacja parametrów + uzasadnienie, dlaczego ta została wybrana jako najważniejsza/najlepsza).

Przygotowanie aplikacji

Aplikacja została przygotowana z wykorzystaniem Pythona i biblioteki PyQt, która umożliwiła utworzenie okna z możliwością ustawiania parametrów, trybów oraz przedstawiania wizualizacji.

Aby uruchomić program należy pobrać kod lub paczkę „.zip” z repozytorium. Repozytorium znajduje się pod linkiem https://github.com/Julnowak/Optymalizacja-wielokryterialna/Projekt, jednak o dostęp należy poprosić autorów kodu. Jeżeli rozmiar plików na to pozwoli, wówczas pliki zostaną załączone również na platformie Upel.

Aby program działał prawidłowo należy po otwarciu w edytorze Pythona zainstalować wszystkie biblioteki zawarte w pliku „requirements.txt” poprzez komendę „pip install requirements.txt”. Gdy wszystkie biblioteki będą już zainstalowane, należy przejść do folderu „GUI” i uruchomić plik „**mainwindow.py**”. Podjęcie tej akcji powinno skutkować pojawieniem się okna aplikacji. W razie problemów, prosimy o kontakt z autorami.



Rys. 1 Zrzut ekranu przedstawiający interfejs

Podsumowanie

- co zostało zrobione,

- jakie wyniki wyszły,

- czy wyniki są satysfakcjonujące, - propozycje tego, co można zrobić, żeby poprawić wyniki lub rozwinąć projekt dalej.