Sprawozdanie

Implementacji algorytmu genetycznego dla problemu plecakowego (nr. 17)

Wojciech Falkowski, nr. indeksu: 18417

Wstęp

W ramach zadania zaprojektowano i zaimplementowano algorytm genetyczny w celu rozwiązania problemu plecakowego, który jest znanym zagadnieniem w dziedzinie badań operacyjnych i informatyki teoretycznej. Problem plecakowy polega na wybraniu zestawu przedmiotów o maksymalnej łącznej wartości, nie przekraczając przy tym określonej maksymalnej wagi.

Cel eksperymentu

Głównym celem eksperymentu była implementacja algorytmu genetycznego, który miał za zadanie znaleźć optymalne lub blisko optymalne rozwiązania problemu plecakowego, biorąc pod uwagę ograniczenia wagi. Maksymalna waga plecaka wynosi 500 jednostek, a przedmiotów możliwych do zapakowania jest 200, które zostały wygenerowane w sposób losowy. Waga przedmiotu jest z przedziału 1–10, a wartość 1–100.

Metodologia

Algorytm genetyczny został zaimplementowany w języku TypeScript z wykorzystaniem środowiska Node.js. Oto kluczowe elementy implementacji:

Reprezentacja osobnika: Każdy osobnik w populacji jest reprezentowany przez ciąg genów (binarny), gdzie każdy gen odpowiada obecności (1) lub nieobecności (0) danego przedmiotu w plecaku.

Funkcje algorytmu: Algorytm składa się z kilku kluczowych funkcji takich jak inicjalizacja populacji, ocena osobników, selekcja, krzyżowanie, mutacja i selekcja naturalna.

Ocena osobników: Każdy osobnik jest oceniany na podstawie sumarycznej wartości przedmiotów, które zawiera, pod warunkiem że łączna waga przedmiotów nie przekracza dopuszczalnej maksymalnej wagi.

Wskaźnik mutacji (mutationRate):

> **Opis:** Wskaźnik mutacji określa prawdopodobieństwo zmiany każdego genu w chromosomie (osobniku). Wysoki wskaźnik mutacji zwiększa różnorodność genetyczną populacji, co może prowadzić do odkrywania nowych, potencjalnie lepszych rozwiązań.
>
> **Zakres wartości:** 0.01 do 0.05

Wpływ na wyniki: Niskie wartości wskaźnika mutacji (0.01) mogą ograniczać eksplorację przestrzeni rozwiązań, co skutkuje mniejszą różnorodnością i potencjalnie gorszymi wynikami. Wyższe wartości wskaźnika mutacji (0.05) zwiększają różnorodność genetyczną, co może prowadzić do lepszych wyników, ale również zwiększa zmienność wyników.

Wskaźnik krzyżowania (crossoverRate):

Opis: Wskaźnik krzyżowania określa prawdopodobieństwo wymiany genów między parami osobników podczas operacji krzyżowania. Wysoki wskaźnik krzyżowania sprzyja tworzeniu nowych kombinacji genów, co może poprawić efektywność algorytmu.

Zakres wartości: 0.7 do 0.9

Wpływ na wyniki: Niższe wartości wskaźnika krzyżowania (0.7) mogą prowadzić do mniejszej zmienności i bardziej stabilnych wyników. Wyższe wartości wskaźnika krzyżowania (0.9) zwiększają zmienność wyników, co może być korzystne dla eksploracji, ale może również prowadzić do większej niestabilności.

Rozmiar populacji (populationSize):

Opis: Rozmiar populacji określa liczbę osobników w każdej generacji. Większa populacja zwiększa różnorodność genetyczną i może prowadzić do bardziej stabilnych wyników, ale wymaga również większej mocy obliczeniowej.

Zakres wartości: 50 do 100

Wpływ na wyniki: Mniejsza populacja (50) może prowadzić do szybszego zbieżności, ale z większym ryzykiem utknięcia w lokalnym optimum. Większa populacja (100) sprzyja lepszej eksploracji przestrzeni rozwiązań i bardziej stabilnym wynikom z mniejszym odchyleniem standardowym.

Implementacja

Kod źródłowy został umieszczony na platformie GitHub i jest dostępny pod adresem:

https://github.com/WojciechxFalkowski/metaheurtystyki-projekt

Implementacja obejmuje wczytywanie danych z pliku, generowanie początkowej populacji, a następnie iteracyjne przetwarzanie populacji w celu znalezienia optymalnych rozwiązań.

Struktura projektu

- node_modules/: Katalog zawierający moduły Node.js zainstalowane przez npm, które są używane w projekcie. Jest to standardowy katalog dla projektów Node.js, gdzie przechowywane są wszystkie zależności.

- .gitignore: Plik konfiguracyjny dla Git, który określa pliki i katalogi ignorowane przez system kontroli wersji. Zawiera zazwyczaj node_modules/ i inne pliki konfiguracyjne, które nie powinny być wersjonowane.

- data.json: Plik danych używany przez algorytm genetyczny. Zawiera informacje o przedmiotach dostępnych do "zapakowania" do plecaka, takie jak ich waga i wartość (200 przedmiotów, waga plecaka [capacity] 500).

- generateData.js: Skrypt JavaScript służący do generowania danych wejściowych, które są następnie zapisywane do pliku data.json. Umożliwia tworzenie różnorodnych scenariuszy testowych dla algorytmu genetycznego.

- main.ts: Główny plik źródłowy projektu napisany w TypeScript, który zawiera logikę implementacji algorytmu genetycznego. To z tego pliku uruchamiana jest cała procedura optymalizacyjna.

- nodemon.json: Plik konfiguracyjny dla Nodemon, narzędzia służącego do automatycznego restartowania projektu Node.js po wykryciu zmian w plikach źródłowych. Ułatwia rozwój aplikacji poprzez szybkie odświeżanie zmian.

- tsconfig.json: Plik konfiguracyjny dla kompilatora TypeScript, określający opcje kompilacji TS do JS, takie jak wersja ECMAScript, ścieżki do plików, itd.

- results.json: Plik wynikowy, w którym zapisywane są wyniki działania algorytmu genetycznego. Każdy wynik reprezentuje potencjalne rozwiązanie problemu plecakowego, z określoną kombinacją przedmiotów, parametrów, ich łączną wartością i wagą.

- analyzeResults.js: Skrypt analizujący wyniki, znajdujący wartości minimalne, maksymalne, średnie, wariancję oraz odchylenie standardowe.

Instrukcja uruchamiania projektu

Aby uruchomić projekt na własnym komputerze, należy zainstalować niezbędne zależności oraz uruchomić poszczególne skrypty. Poniżej znajduje się szczegółowy opis kroków wymaganych do uruchomienia projektu, zakładając że masz już zainstalowany Node.js w wersji 20.11.1.

Wstępne ustawienia

Klonowanie repozytorium lub pobranie projektu

git clone https://github.com/WojciechxFalkowski/metaheurtystyki-projekt

Instalacja zależności

Otwórz terminal lub wiersz poleceń i przejdź do katalogu głównego projektu, gdzie znajduje się plik package.json.

Wykonaj komendę npm install w terminalu. Polecenie to zainstaluje wszystkie zależności wymienione w pliku package.json, które są niezbędne do działania projektu.

Uruchomienie analizy wyników

Aby uruchomić skrypt analizujący wyniki zapisane w pliku results.json, użyj polecenia

node ./analyzeResults.js

Upewnij się, że w katalogu projektu znajduje się plik results.json z odpowiednimi danymi.

Skrypt analyzeResults.js powinien wyświetlić wyniki analizy w konsoli.

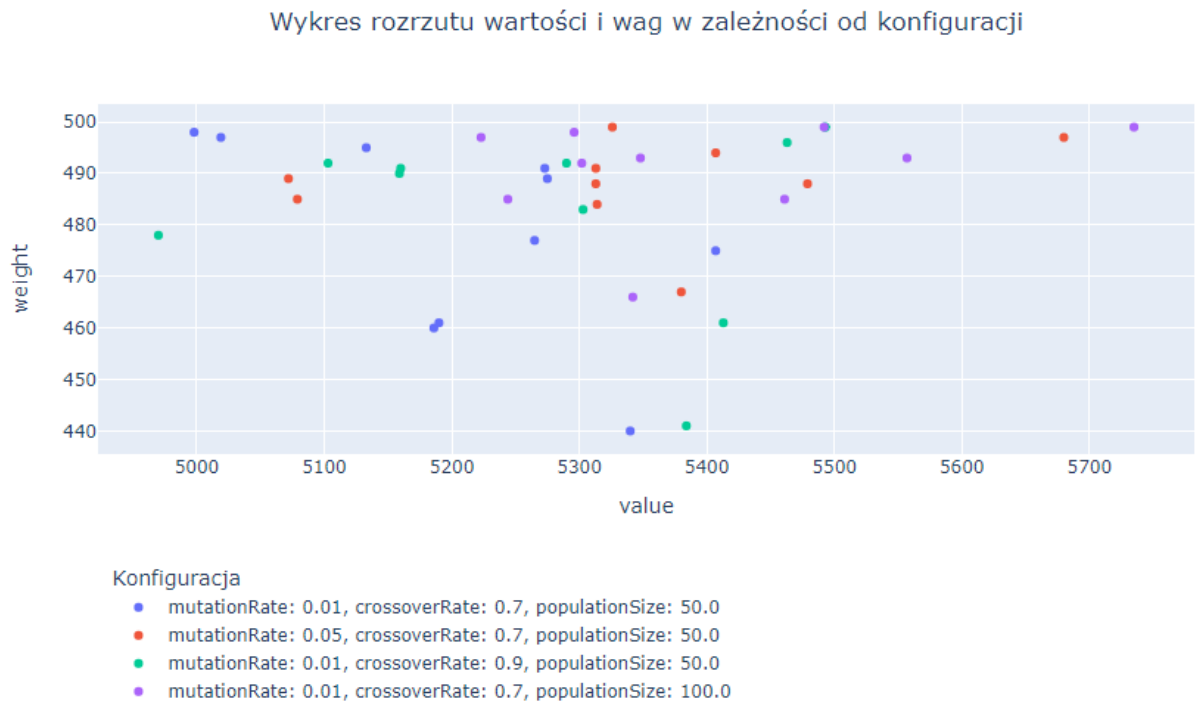
Uruchomienie głównego skryptu algorytmu genetycznego

Użyj npx nodemon ./main.ts do uruchomienia głównego skryptu projektu.

Interpretacja wyników

Dla każdej konfiguracji parametrów przeprowadzono 10 symulacji, a wyniki przedstawiono w formie tabelarycznej i graficznej. Poniżej znajdują się szczegółowe wyniki dla każdej konfiguracji:

Konfiguracja	Średnia wartość	Odchylenie standardowe wartości	Średnia waga	Odchylenie standardowe wagi
mutationRate: 0.01, crossoverRate: 0.7, populationSize: 50	5208.6	124.59	478.3	18.46
mutationRate: 0.05, crossoverRate: 0.7, populationSize: 50	5336.3	168.27	488.2	8.45
mutationRate: 0.01, crossoverRate: 0.9, populationSize: 50	5273.8	162.5	482.3	17.23
mutationRate: 0.01, crossoverRate: 0.7, populationSize: 100	5400	152.06	490.7	9.58



1. Średnia wartość rozwiązania:

- Najwyższą średnią wartość uzyskano dla konfiguracji z mutationRate: 0.05, crossoverRate: 0.7, populationSize: 50 (5336.30). Wysokie prawdopodobieństwo mutacji pozwala na większą różnorodność genetyczną, co może prowadzić do lepszych rozwiązań.
- Najniższą średnią wartość uzyskano dla konfiguracji z mutationRate: 0.01, crossoverRate: 0.7, populationSize: 50 (5208.60). Niskie prawdopodobieństwo mutacji może ograniczać eksplorację przestrzeni rozwiązań, co skutkuje gorszymi wynikami.

2. Odchylenie standardowe wartości:

- Najniższe odchylenie standardowe wartości uzyskano dla konfiguracji z mutationRate: 0.01, crossoverRate: 0.7, populationSize: 100 (152.06). Większa populacja może stabilizować wyniki, redukując ich zmienność.
- Najwyższe odchylenie standardowe wartości uzyskano dla konfiguracji z mutationRate: 0.05, crossoverRate: 0.7, populationSize: 50 (168.27). Większa różnorodność genetyczna prowadzi do większej zmienności wyników.

3. Średnia waga rozwiązania:

- Najwyższą średnią wagę uzyskano dla konfiguracji z mutationRate: 0.01, crossoverRate: 0.7, populationSize: 100 (490.70). Większa populacja może pomóc w znalezieniu bardziej zbalansowanych rozwiązań.
- Najniższą średnią wagę uzyskano dla konfiguracji z mutationRate: 0.01, crossoverRate: 0.7, populationSize: 50 (478.30).

4. Odchylenie standardowe wagi:

- Najniższe odchylenie standardowe wagi uzyskano dla konfiguracji z mutationRate: 0.05, crossoverRate: 0.7, populationSize: 50 (8.45). Stabilność wag może wskazywać na skuteczniejszą selekcję genów.
- Najwyższe odchylenie standardowe wagi uzyskano dla konfiguracji z mutationRate: 0.01, crossoverRate: 0.9, populationSize: 50 (17.23). Wysokie prawdopodobieństwo krzyżowania zwiększa zmienność wyników.

Wnioski

Wielokrotne symulacje potwierdzają, że algorytm genetyczny jest skutecznym narzędziem do rozwiązywania tego typu problemów, jednak odpowiedni dobór parametrów jest kluczowy dla uzyskania najlepszych wyników.

Na podstawie przeprowadzonych symulacji, najlepsze wyniki uzyskano dla konfiguracji mutationRate: 0.01, crossoverRate: 0.7, populationSize: 100. Ta konfiguracja osiągnęła najwyższą średnią wartość rozwiązania (5400.00) przy niskim odchyleniu standardowym (152.06), co świadczy o stabilności i wysokiej jakości uzyskanych rozwiązań. Dodatkowo, średnia waga (490.70) i niskie odchylenie standardowe wagi (9.58) wskazują na równowagę między wartością a wagą przedmiotów w plecaku. W związku z tym, konfiguracja ta jest rekomendowana jako optymalna dla rozwiązania problemu plecakowego za pomocą algorytmu genetycznego.