Projektowanie Efektywnych Algorytmów

Projekt

23/01/2024

263896 Patryk Jurkiewicz

(7) Algorytm mrówkowy

1. Sformułowanie zadania

Celem zadania jest eksploracja i praktyczna implementacja Algorytmu mrówkowego w kontekście rozwiązania złożonego problemu komiwojażera. Algorytm mrówkowy, czerpiący inspirację z naturalnych zachowań mrówek podczas poszukiwania pożywienia, stanowi potężną technikę optymalizacyjną, bazującą na zasadach inteligencji roju. W teorii algorytm ten wykorzystuje fascynującą zdolność mrówek do pozostawiania feromonów w śladzie, co z kolei wpływa na decyzje podjęte przez kolejne mrówki. W kontekście problemu komiwojażera, mrówki są modelowane jako agenci poruszający się między miastami w celu stworzenia optymalnych tras. Feromony, które pozostawiają w śladzie, stanowią swoiste znaczniki jakości tras, wpływając na preferencje i wybory kierunków dla kolejnych mrówek. Kluczowymi parametrami, determinującymi skuteczność algorytmu, są: α i β, które regulują balans pomiędzy wpływem feromonów a widocznością ścieżki; ρ, który kontroluje proces ewaporacji feromonów; m, liczba mrówek uczestniczących w poszukiwaniach; $\tau 0$, wartość początkowa feromonu, oraz heurystyka visibility, opierająca się na odległości między poszczególnymi miastami. Przy opracowywaniu algorytmu, istotne jest także uwzględnienie założeń projektowych, w tym ustalania błędu procentowego w zależności od rozmiaru instancji problemu komiwojażera. Zadaniem algorytmu jest efektywne przeszukiwanie przestrzeni rozwiązań, generowanie optymalnych tras oraz adaptacja do zmiennych warunków, co sprawia, że Algorytm mrówkowy staje się potężnym narzędziem w rozwiązywaniu problemów optymalizacyjnych.

2. Metoda

Algorytm mrówkowy, w kontekście rozwiązania problemu komiwojażera, wykorzystuje nowatorską heurystykę znalezioną w metodzie QAS (Quasi-ant System). Metoda QAS wprowadza dodatkowe usprawnienia, które zwiększają efektywność algorytmu poprzez bardziej precyzyjne zarządzanie feromonami i wybieranie tras. W funkcji `initializePheromone`, feromony początkowe są inicjalizowane wartościami odwrotnymi do długości tras wygenerowanych przez mrówki przy użyciu algorytmu najbliższego sąsiada. W metodzie QAS, proces wyboru kolejnego miasta przez mrówkę, zaimplementowany w funkcji `selectNextCity`, podlega dodatkowej modyfikacji. Dodatkowo do wpływu feromonów i widoczności, wprowadza się wartość atrakcyjności, która uwzględnia różnice między aktualnym poziomem feromonu a początkowym. To usprawnienie pozwala na bardziej wyważone decyzje mrówek, co wpływa korzystnie na jakość tras. Główna funkcja algorytmu, `Main`, obejmuje pętlę, w której mrówki poruszają się między miastami. W metodzie QAS, feromony są aktualizowane zgodnie z wyważonymi wartościami atrakcyjności i widoczności, co zwiększa szanse na optymalne wybory tras. Parametry α, β, ρ, m, τ0, oraz heurystyka `visibility` są skrupulatnie dostosowywane, by wprowadzić optymalne ustawienia dla metody QAS. Warto podkreślić, że całość algorytmu mrówkowego z wykorzystaniem metody QAS, jak opisano powyżej, stanowi zaawansowaną technikę optymalizacyjną, pozwalającą na skuteczne rozwiązanie problemu komiwojażera przy minimalizacji błędu procentowego, zwłaszcza dla rozmiarów instancji O(24ln(n)).

Początkowa ilość feromonów na każdej krawędzi wynosi:

$$\tau_0 = \frac{m}{C^{nn}}$$

3. Algorytm

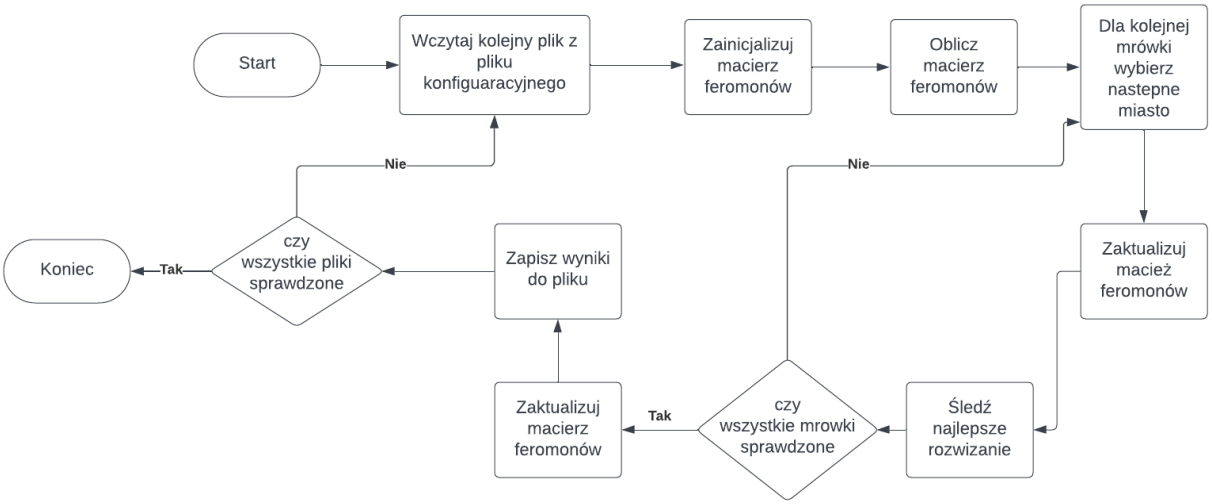
Opisany algorytm mrówkowy QAS, mający na celu rozwiązanie problemu komiwojażera, składa się z kilku kluczowych kroków, które precyzyjnie kontrolują poszukiwania optymalnych tras przez mrówki.

Część 1: Inicjalizacja feromonów (initializePheromone): Na początku algorytmu feromony są inicjalizowane. Dla każdej z mrówek generowana jest losowa trasa, a feromony na trasie są ustawiane w sposób odwrotnie proporcjonalny do długości tej trasy. Następnie, wartości feromonów na wszystkich trasach są znormalizowane, zapewniając równomierne rozpoczęcie poszukiwań.

Część 2: Obliczanie widoczności między miastami (calculateVisibility): Algorytm uwzględnia widoczność między każdą parą miast, gdzie widoczność jest zdefiniowana jako odwrotność odległości między miastami. To uwzględnienie wpływa na preferencje mrówek podczas wyboru tras.

Część 3: Wybór kolejnego miasta przez mrówkę (selectNextCity): Proces ten polega na ocenie prawdopodobieństwa wyboru kolejnego miasta przez mrówkę. Prawdopodobieństwo to jest obliczane na podstawie wpływu feromonów i widoczności ścieżki. Wartości parametrów α i β regulują wagę wpływu feromonów i widoczności, a proces ten jest oparty na losowaniu, co wprowadza element losowości w wybór tras.

Część 4: Optymalizacja kolonii mrówek (antColonyOptimization): Główna pętla algorytmu, zwana AOMain, prowadzi mrówki przez iteracje, podczas których przemieszczają się między miastami. Mrówki pozostawiają feromony na trasach, a ich wybory są korygowane w oparciu o atrakcyjność tras. Wartości parametrów α, β, ρ, m, τ0 oraz heurystyka visibility są dostosowywane, by zoptymalizować działanie algorytmu podczas każdej iteracji.



Rys 1. Schemat blokowy algorytmu

4. Dane testowe

W celu przetestowania i dostrojenia naszego algorytmu wybraliśmy zestaw konkretnych instancji problemu komiwojażera. Te instancje posłużą nam do oceny skuteczności i poprawności działania algorytmu oraz do określenia odpowiednich parametrów.

Do wykonania badań wybrano następujący zestaw instancji ze stron:
http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/
http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/XML-TSPLIB/instances/

burma14.txt

fri26.txt

ft53.txt

ftv70.txt

gr96.txt

kroA100.txt

kroB150.txt

kroB200.txt

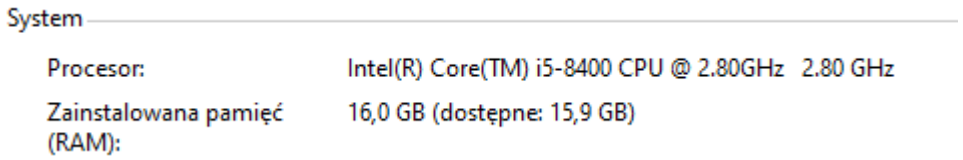
rbg323.txt

5. Procedura badawcza

Aby wykonać ten proces, należy wypełnić plik ini liniami z nazwami plików z danymi oraz ilością, ile razy mają zostać powtórzone. Później, należy uruchomić plik wykonywalny (exe). Wyniki zostaną zapisane w pliku "wyjscie.txt" w formacie, który przypomina strukturę przedstawioną w poniżej:

Plik: nazwa_pliku
Wynik: Długość najlepszej trasy
Czas wykonania: czas

Program był testowany na komputerze o tej specyfikacji:



W kodzie używane są biblioteki `<chrono>` oraz `<ctime>` do pomiaru czasu wykonania algorytmu. Klasa `std::chrono::high_resolution_clock` dostarcza dostęp do najwyższej dostępnej rozdzielczości zegara czasu rzeczywistego na danym systemie. Początkowy czas (`timer_start`) jest reprezentowany jako punkt w czasie, a różnica między tym punktem a aktualnym czasem daje czas wykonania algorytmu. Warto zauważyć, że `chrono::duration_cast` jest używane do konwersji różnicy czasów na milisekundy, co pozwala uzyskać bardziej czytelną jednostkę czasu.

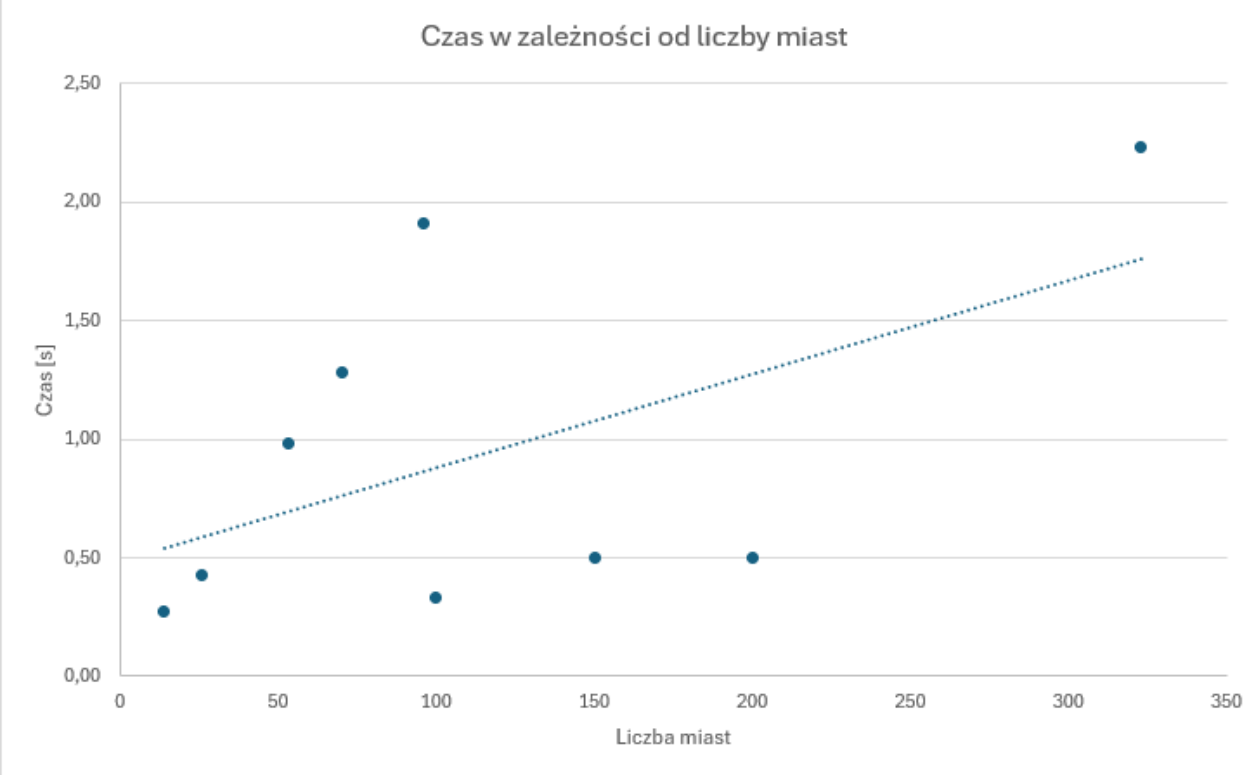
Poniżej znajduje się odpowiedni fragment kodu, który mierzy czas wykonania algorytmu:

```
auto start_time =
chrono::high_resolution_clock::now();


// ... Reszta algorytmu ...


auto end_time =
chrono::high_resolution_clock::now();

elapsedTime = chrono::duration<double>(end_time -
start_time).count();
```

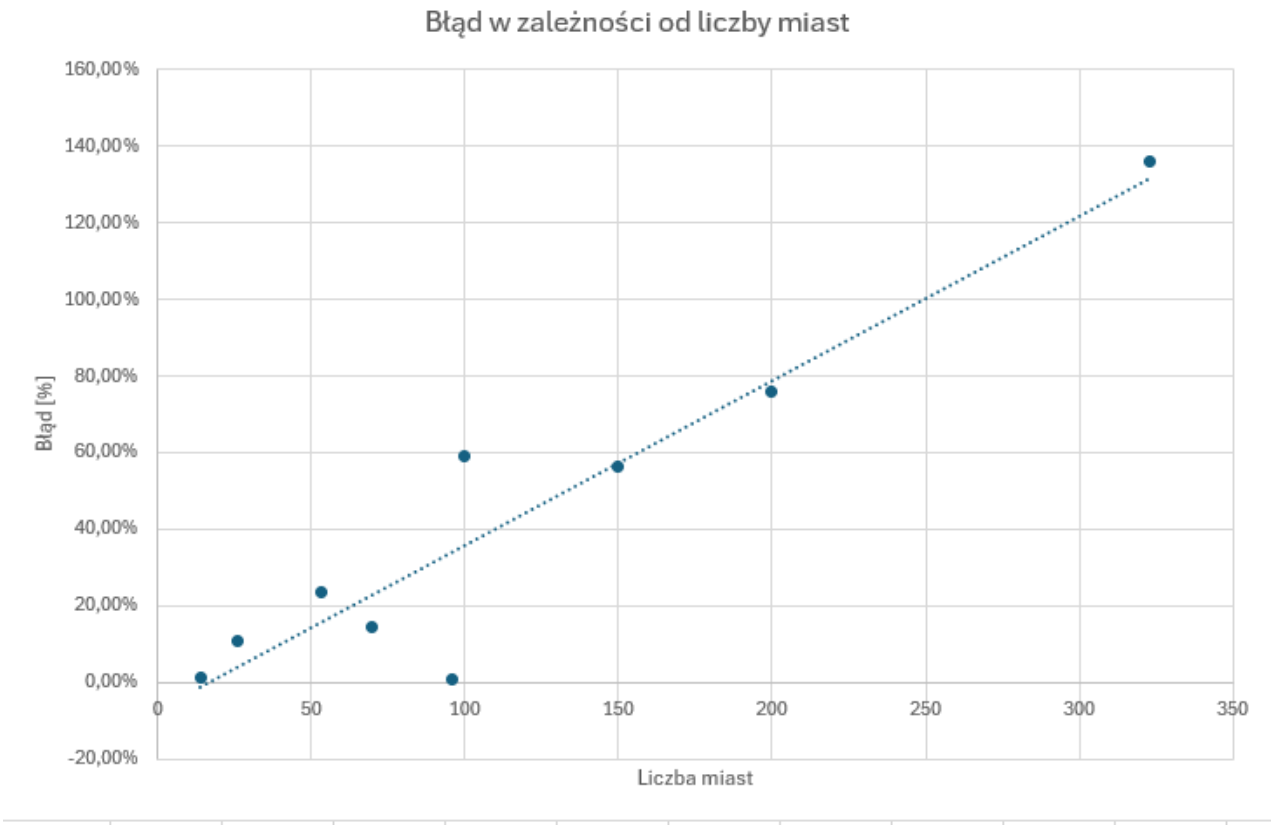
6. Wyniki

Miasta	Koszt	Czas	Optimum	Błąd
14	3278	0,27	3323	1,35%
26	1036	0,43	937	10,57%
53	8533	0,98	6905	23,58%
70	2231	1,28	1950	14,41%
96	55706	1,91	55209	0,90%
100	33897	0,33	21282	59,28%
150	40857	0,50	26130	56,36%
200	51780	0,50	29437	75,90%
323	3133	2,23	1326	136,27%

Rys 2. Tabela z wynikami



Rys 3. Wykres czasu od miast



Rys 4. Wykres błędu od miast

7. Analiza wyników i wnioski

W trakcie analizy algorytmu mrówkowego w kontekście jego porównania z algorytmem Held-Karpa, Tabu Search i Brute Force, ujawniły się różnice zarówno w zakresie czasu wykonania, jak i precyzji osiąganych wyników. Warto zauważyć, że przyjęte zmienne sprawdziły się w pewnym stopniu, podkreślając zarówno mocne strony, jak i wyzwania związane z tym algorytmem. Pod względem zużycia pamięciowego, algorytm mrówkowy okazał się bardziej efektywny niż Brute Force, co jest zgodne z przewidywaniami. Jego zdolność do skuteczniejszej eksploracji przestrzeni rozwiązań przekładała się na mniejsze zapotrzebowanie na zasoby pamięciowe. Jednakże, chociaż czas wykonania algorytmu mrówkowego był najszybszy w porównaniu z innymi algorytmami, nie przekładało się to bezpośrednio na osiągnięcie najbardziej precyzyjnych wyników. Wyniki analizy wskazały, że algorytm mrówkowy wykazywał mniej dokładność w porównaniu z optymalnymi rozwiązaniami. Szczególnie dla zbioru danych obejmującego 323 miasta, zaobserwowano znaczący błąd wynoszący 136,27%. To sugeruje, że algorytm może napotykać trudności w przypadku bardziej złożonych instancji problemu komiwojażera, być może ze względu na probabilistyczne podejście, które wprowadza pewne niestabilności. W kontekście ogólnej wydajności, wyniki sugerują, że algorytm mrówkowy wypadał najgorzej spośród porównywanych metod. Wskazuje to na konieczność uwzględnienia jego probabilistycznego charakteru oraz wpływu parametrów na jakość uzyskiwanych rozwiązań. Podejście do dostosowania parametrów i ewolucji strategii feromonów może być kluczowe dla poprawy precyzji i stabilności algorytmu. Warto podkreślić, że prezentowane wyniki stanowią punkt wyjścia do dalszych badań. Eksperymenty z różnymi zestawami parametrów mogą prowadzić do optymalizacji algorytmu mrówkowego, szczególnie w kontekście bardziej złożonych problemów. Zmiany i dostosowania w strategii ewolucji feromonów mogą być kluczowe dla uzyskania bardziej precyzyjnych rezultatów.