**WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI**

**POLITECHNIKA WROCŁAWSKA**

KIERUNEK: INFORMATYKA TECHNICZNA

**Projektowanie efektywnych algorytmów**

**PROJEKT** 3

**Autor**: Maksymilian Łukaszewski 272975

**Prowadzący**: dr inż. Dariusz Banasiak

Czwartek 7:30

1. **Cel Projektu**

Celem projektu było zaprojektowanie i zaimplementowanie algorytmu mrówkoego dla asymetrycznego problemu komiwojażera. Dodatkowo należało zbadać i przeanalizować czas potrzebny na rozwiązanie problemu w zależności od następujących parametrów:

Ilość mrówek

Ilość zostawianych feromonów

Współczynnik wyparowywania feromonów

Czas działania

**Przyjęte założenia**

Podczas projektowania i implementacji przyjęto następujące założenia:

* zadanie zrealizowane jest w ramach jednego programu konsolowego
* wszystkie struktury danych alokowane są dynamicznie
* odległości pomiędzy miastami są liczbami całkowitymi
* program umożliwia wczytanie danych wejściowych z pliku
* program jest stworzony zgodnie z zasadami programowania obiektowego
* zaimplementowano odcięcie „czasowe” górne

**Język Programowania i środowisko**

Program, który stworzono w ramach projektu został napisany w języku C++. Badania przeprowadzono na komputerze lenovo Thinkpad T480 z systemem operacyjnym Windows10. Badania przeprowadzono po kompilacji i buildowaniu kodu w trybie *debug.* Podczas przeprowadzania testów wyłączono wszystkie procesy w tle, w celu uzyskania jak najbardziej precyzyjnych wyników.

1. **Wstęp teoretyczny: Algorytm Mrówkowy (ACO)**

Algorytm mrówkowy (Ant Colony Optimization, ACO) to metaheurystyka inspirowana zachowaniem mrówek szukających najkrótszej drogi do pożywienia. Sztuczne mrówki poruszają się po grafie, kierując się śladem feromonowym oraz heurystyką (np. odległością). Po każdej iteracji feromony odparowują i są wzmacniane na korzystnych trasach, co sprzyja znajdowaniu optymalnych rozwiązań.

Algorytm balansuje między **eksploracją** (badaniem nowych tras) a **eksploatacją** (wybieraniem sprawdzonych ścieżek), co pozwala unikać lokalnych minimów. Dzięki adaptacyjności i równoległemu przeszukiwaniu przestrzeni rozwiązań znajduje zastosowanie w optymalizacji tras, harmonogramowaniu i zarządzaniu sieciami.

1. **Metodologia badania**

Eskperyment miał sprawdzić efektywność algorytmów w zależności od zadanych im parametrów. Badania przeprowadzono dla 2 reprezentatywnych ilości miast:

55 w pliku ftv55.atsp

Na podstawie wstępnych badań ustalono następujące parametry bazowe:

-ilość mrówek = 40

-ilość zostawianego feromonu = 100

-ilość wyparowywanego feromonu = 0.1

-górna granica czasowa 30s

Następnie wykonano po 9 pomiarów, jeden z bazowymi wartościami i po 2 ze zwiększonym i zmniejszonym parametrem dwukrotnie (czyli zmieniano ilość mrówek na 20/80, bez zmiany pozostałych parametrów, dokonywano pomiarów i analogicznie z pozostałymi parametrami). Nie badano wpływu współczynników alfa i beta, więc do wszystkich badań użyto alpha = 1.0 i beta = 2.0.

**4. Algorytm Mrówkowy (Ant Colony Optimization, ACO)**

Algorytm mrówkowy (ACO) to metaheurystyczna metoda optymalizacji inspirowana zachowaniem kolonii mrówek podczas poszukiwania najkrótszej ścieżki do źródła pożywienia. Wykorzystuje mechanizm śladów feromonowych, które prowadzą do coraz lepszych rozwiązań w kolejnych iteracjach.

W ACO wiele mrówek jednocześnie konstruuje trasy, wybierając kolejne kroki na podstawie kombinacji feromonów (śladów pozostawionych przez inne mrówki) oraz heurystyki (np. odwrotności odległości). Po zakończeniu iteracji najlepsze trasy są wzmacniane dodatkowymi feromonami, co zwiększa prawdopodobieństwo wyboru tych ścieżek w przyszłości.

**Podstawowe etapy algorytmu ACO:**

1. Inicjalizacja:
   * Ustal liczbę mrówek oraz parametry algorytmu (wpływ feromonów, waga heurystyki, współczynnik parowania feromonów).
   * Zainicjalizuj poziom feromonów na wszystkich krawędziach.
2. Konstrukcja tras:
   * Każda mrówka zaczyna w losowym mieście.
   * Wybiera kolejne miasto z prawdopodobieństwem zależnym od poziomu feromonów i odległości.
   * Po odwiedzeniu wszystkich miast powraca do miasta startowego.
3. Aktualizacja feromonów:
   * Wszystkie feromony ulegają parowaniu (odparowaniu), aby zapobiec ich nieskończonemu wzrostowi.
   * Mrówki wzmacniają feromonami trasy proporcjonalnie do ich jakości (im krótsza trasa, tym więcej feromonów).
4. Kryterium stopu:
   * Algorytm kończy działanie po osiągnięciu maksymalnej liczby iteracji lub upływie określonego czasu.

ACO jest szeroko stosowane w problemach kombinatorycznych, takich jak problem komiwojażera (TSP), planowanie tras czy optymalizacja sieci. Dzięki mechanizmowi samoadaptacji i współpracy wielu mrówek, algorytm skutecznie unika lokalnych minimów i może znaleźć wysokiej jakościrozwiązania w rozsądnym czasie.

1. **Metodologia badania**

**Klasy Pomocnicze**

**5.1. Klasa Matrix**

**Konstruktor i Destruktor**

* **Matrix(int size)**: Inicjalizuje macierz o zadanym rozmiarze.
* **~Matrix()**: Zwalnia pamięć zaalokowaną dla macierzy.

**Operacje na macierzy**

* **freeMatrix()**: Zwalnia pamięć zaalokowaną dla macierzy.
* **loadFromFile(const string& filename)**: Ładuje macierz z pliku. Odczytuje rozmiar macierzy i jej elementy.
* **generateRandom(int newSize, int min, int max)**: Generuje losową macierz o określonym rozmiarze, z wartościami w podanym zakresie (min, max).
* **print()**: Wypisuje macierz na ekran z sformatowanym wyjściem.
* **getSize()**: Zwraca rozmiar macierzy.
* **getMatrix()**: Zwraca wskaźnik na surową tablicę macierzy.
* **getValue(int row, int col)**: Zwraca wartość z macierzy na podanych (wiersz, kolumna).

**5.2. Klasa Timer**

**Operacje na timerze**

* **Timer()**: Konstruktor inicjalizuje elapsedTime na 0.
* **start()**: Rozpoczyna pomiar czasu, zapisując czas początkowy.
* **stop()**: Zatrzymuje pomiar czasu, zapisując czas końcowy i obliczając czas wykonania w
* milisekundach.
* **getElapsed()**: Zwraca czas wykonania.
* \**printElapsed(const char* label)\*\*: Wypisuje czas wykonania z etykietą.

**6. Algorytm Mrówkowy (Ant Colony Optimization, ACO)**

**6.1 Opis teoretyczny**

Algorytm mrówkowy (Ant Colony Optimization, ACO) to metaheurystyczna metoda optymalizacji inspirowana zachowaniem rzeczywistych mrówek podczas poszukiwania optymalnych tras między gniazdem a źródłem pożywienia. W ACO mrówki poruszają się po grafie problemu, stopniowo wzmacniając najlepsze trasy za pomocą feromonów, co prowadzi do znalezienia coraz lepszych rozwiązań.

Kluczowe elementy algorytmu mrówkowego:

* Feromony – każda krawędź w grafie ma przypisaną wartość feromonów, która jest aktualizowana po każdej iteracji.
* Funkcja wyboru ścieżki – decyzje o wyborze kolejnej krawędzi opierają się na poziomie feromonów oraz heurystyce (np. odwrotności odległości).
* Parowanie feromonów – feromony stopniowo odparowują, co zapobiega ich nadmiernej koncentracji na pojedynczych ścieżkach i umożliwia eksplorację nowych tras.
* Aktualizacja feromonów – po każdej iteracji mrówki wzmacniają feromonami znalezione trasy, a najlepsze rozwiązania pozostają preferowane w kolejnych iteracjach.

Algorytm mrówkowy jest szeroko stosowany w problemach kombinatorycznych, takich jak problem komiwojażera (TSP), planowanie tras czy optymalizacja sieci.

**6.2 Złożoność obliczeniowa**

Złożoność obliczeniowa ACO zależy od kilku czynników:

* Liczba mrówek (m) – więcej mrówek oznacza większą eksplorację, ale także większy koszt obliczeniowy.
* Liczba miast (n) – każda mrówka konstruuje pełną trasę, co prowadzi do złożoności O(m⋅n²) w każdej iteracji.
* Liczba iteracji (k) – całkowita złożoność zależy od krotności powtórzeń algorytmu, co daje O(k⋅m⋅n²).

W praktyce dobór odpowiednich parametrów (liczba mrówek, współczynnik parowania feromonów) pozwala na uzyskanie wysokiej efektywności algorytmu w stosunku do innych metod optymalizacji heurystycznej.

**6.3 Opis implementacji**

Poniżej przedstawiono sposób działania algorytmu mrówkowego w implementacji:

1. Inicjalizacja:
   * Tworzona jest kolonia mrówek o określonej liczbie osobników.
   * Ustalane są parametry algorytmu, takie jak ilość pozostawianego feromonu, współczynnik parowania oraz maksymalny czas działania algorytmu.
   * Macierz feromonów jest inicjalizowana – początkowo każda krawędź ma jednakowy poziom feromonów (=1).
2. Konstrukcja tras:
   * Każda mrówka rozpoczyna swoją podróż z wybranego miasta startowego.
   * Następnie wybiera kolejne miasta zgodnie z regułą prawdopodobieństwa, która uwzględnia zarówno poziom feromonów, jak i heurystykę opartą na odwrotności odległości.
   * Proces powtarza się, aż każda mrówka odwiedzi wszystkie miasta.
   * Po zakończeniu trasy mrówka powraca do miasta startowego, zamykając cykl.
3. Aktualizacja feromonów:
   * Po każdej iteracji następuje proces parowania feromonów – poziom feromonów na wszystkich krawędziach jest zmniejszany zgodnie z ustalonym współczynnikiem parowania.
   * Następnie feromony są wzmacniane przez mrówki, które ukończyły trasę – im krótsza trasa, tym większy przyrost feromonów na jej krawędziach.
4. Kryterium zatrzymania:
   * Algorytm kończy działanie po osiągnięciu limitu czasowego.
5. Wynik:
   * Po zakończeniu algorytmu zwracana jest najlepsza znaleziona trasa oraz jej koszt.

ACO jest efektywną metodą optymalizacji, szczególnie w problemach, gdzie konieczna jest eksploracja wielu różnych ścieżek jednocześnie. Dzięki mechanizmowi feromonów algorytm znajduje wysokiej jakości rozwiązania w rozsądnym czasie.

**7.WYNIKI**

Wyniki pomiarów dla n =55

A close-up of numbers

Description automatically generated

**A close-up of numbers

Description automatically generated**

A close-up of numbers

Description automatically generated

A graph of different colored lines

Description automatically generated

A graph of different colored lines

Description automatically generated

A graph of different colored lines

Description automatically generated

Wyniki dla pomiarów dla n=170

A close up of numbers

Description automatically generated

A close up of numbers

Description automatically generated

A screenshot of a computer

Description automatically generated

A graph with lines and numbers

Description automatically generated

A graph with lines and numbers

Description automatically generated

A graph of a number of numbers and a line graph

Description automatically generated with medium confidence

Wyniki dla n =358

A close up of numbers

Description automatically generated

A screenshot of a calculator

Description automatically generated

A close up of numbers

Description automatically generated

A graph with blue and orange lines

Description automatically generated

A graph with lines and numbers

Description automatically generated

A graph with numbers and lines

Description automatically generated

Zestawienie wyników dla różnych wielkości grafu

**A screenshot of a calculator

Description automatically generated**

**A graph with lines and numbers

Description automatically generated**

**8.WNIOSKI**

Wpływ parametrów na działanie algorytmu mrówkowego jest najbardziej zauważalny dla grafu o rozmiarze n=358. Może to wynikać z faktu, że dla mniejszych grafów algorytm szybko znajduje bardzo dobre rozwiązania, natomiast dla największego rozmiaru dopiero po około 15 sekundach udaje się osiągnąć satysfakcjonujący wynik.

Analiza wpływu poszczególnych parametrów wykazała, że:

* Wyższy współczynnik parowania powoduje mniejszą różnicę błędu w czasie, co wskazuje na stabilizację wyników.
* Zwiększenie ilości feromonu poprawia skuteczność algorytmu, prowadząc do lepszych rozwiązań.
* Większa liczba mrówek negatywnie wpływa na działanie algorytmu w przypadku dużych instancji problemu, co może wynikać ze wzrostu złożoności obliczeniowej i większego nakładu na aktualizację śladów feromonowych.

**9.DODATEK**

**Przykład działania algorytmu mrówkowego (ACO) dla problemu komiwojażera**

**Dane wejściowe:**

| Miasto | A | B | C | D |
| --- | --- | --- | --- | --- |
| A | 0 | 10 | 15 | 20 |
| B | 10 | 0 | 30 | 25 |
| C | 15 | 30 | 0 | 35 |
| D | 20 | 25 | 35 | 0 |

**Parametry algorytmu:**

* Liczba mrówek: 5
* Współczynnik odparowywania feromonów (ρ): 0.5
* Współczynnik wpływu feromonów (α): 1.0
* Współczynnik wpływu heurystyki (β): 2.0
* Początkowa ilość feromonów: 1.0
* Liczba iteracji: 100

**Krok 1: Inicjalizacja**

Każda mrówka losowo wybiera miasto startowe. Załóżmy, że mrówka 1 zaczyna w C i konstruuje trasę:  
C → A → D → B → C  
Koszt trasy: 15 + 20 + 25 + 30 = 90

**Krok 2: Budowanie tras**

Mrówki wybierają kolejne miasta na podstawie prawdopodobieństwa:

A black text on a white background

Description automatically generated

Gdzie:

* τ(i,j) – poziom feromonów na krawędzi (i, j)
* η(i,j) = 1/d(i,j) – heurystyka (odwrotność odległości)

Załóżmy, że po kilku iteracjach najlepsza mrówka utworzyła trasę:  
C → B → A → D → C  
Koszt trasy: 30 + 10 + 20 + 35 = 95

**Krok 3: Aktualizacja feromonów**

Feromony na krawędziach tras są aktualizowane według wzoru:

A black text on a white background

Description automatically generated

Gdzie:

A math equation with black text

Description automatically generated

(Q – stała, L – długość trasy danej mrówki)

Jeśli trasa C → B → A → D → C jest najlepsza, to feromony na jej krawędziach zostaną wzmocnione.

**Krok 4: Kolejne iteracje**

Mrówki tworzą nowe trasy, a algorytm stopniowo koncentruje się na najlepszych ścieżkach.

Po 100 iteracjach algorytm znajduje najlepszą trasę:  
C → A → B → D → C  
Koszt: 15 + 10 + 25 + 35 = 85

**Wynik końcowy:**

* Najlepsza trasa: C → A → B → D → C
* Najniższy koszt: 85
* Liczba iteracji: 100
* Końcowe wartości feromonów wskazują optymalną trasę

Dzięki mechanizmowi feromonów i heurystyki, algorytm skutecznie znalazł krótszą trasę niż początkowe rozwiązania.