MM.P5 Najkrótsza ścieżka w grafie ważonym – ACO

Joanna Sokołowska
Rafał Uzarowicz

Zaimplementować algorytm mrówkowy - ACO (Ant Colony Optimization) do wyznaczania najkrótszej ścieżki w grafie ważonym od punktu A do B. Porównać działanie algorytmu z przeszukiwaniem brute force. WE: plik z listą krawędzi grafu (punkt początkowy, końcowy i waga), punkt startowy i docelowy. WY: najkrótsza ścieżka od punktu A do punktu B i jej koszt

Przyjęte założenia:

1. Mrówki poruszając się po grafie nie wracają do odwiedzonych już wierzchołków, w związku z tym nie każda mrówka w danej iteracji znajdzie ścieżkę.
2. Algorytm dostosowuje się do wyników generowanych przez mrówki poprzez zmianę parametrów balansujących wpływ feromonów i długości krawędzi na generowane ścieżki.
3. Jako wynik zwracana jest najlepsza ze ścieżek znalezionych w trakcie całego działania algorytmu.

Podział obowiązków:

Joanna: akcje dodatkowe, generacja grafów do testów, algorytm brute force, interfejs linii poleceń, klasy graph i vertex, enkapsulacja w klasie ogólnej
Rafał: generacja rozwiązań, aktualizacja feromonów, wczytywanie grafów z pliku, algorytm dijkstry, klasy Ant i Anthill, testy wydajnościowe

Opis algorytmu i architektury:

W działaniu algorytmu ACO wyróżniamy 3 fazy: generowanie rozwiązań, akcje pomocnicze i aktualizacja feromonów.

Generowanie rozwiązań polega na tym, że mrówki wychodzące z wierzchołka początkowego poruszają się po grafie szukając wierzchołka końcowego. Mrówka dokonuje wyboru następnej krawędzi w sposób losowy (z zbioru rozważanych krawędzi wyłączone są te, które prowadzą do odwiedzonych wierzchołków, zgodnie z założeniem). Prawdopodobieństwo wyboru krawędzi jest tym większe, im więcej jest zgromadzonych na niej feromonów i tym mniejsze im większa jest jej długość. Wpływ tych czynników jest regulowany

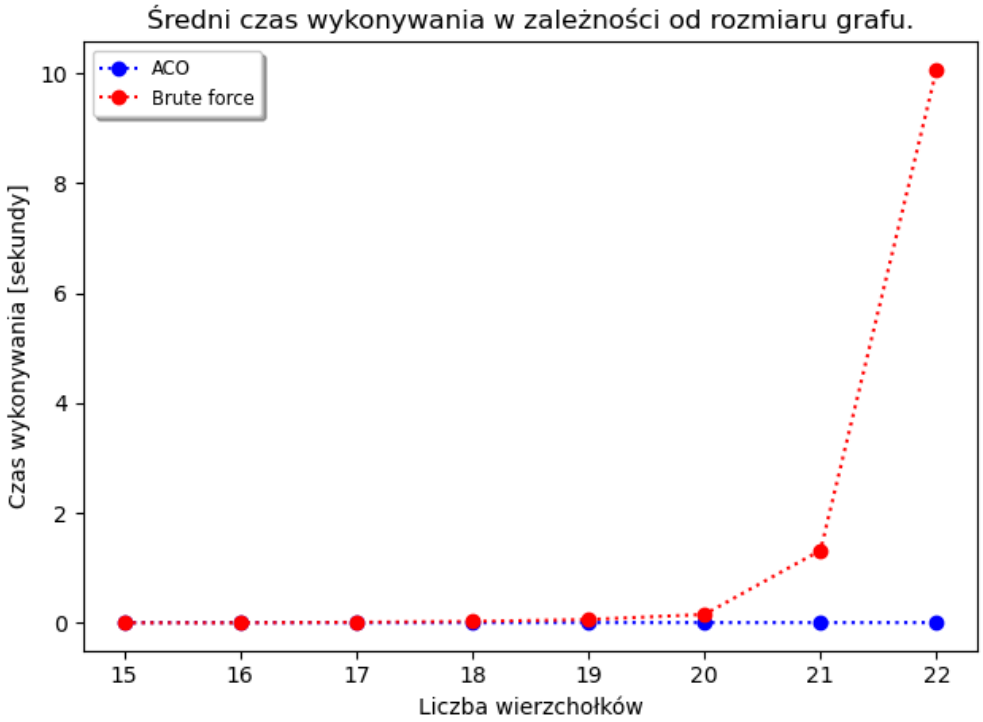
W fazie akcji pomocniczych, jeśli ustawiona została odpowiednia flaga dokonuje się przeszukanie lokalne przestrzeni znalezionych rozwiązań i wybór tych mrówek, które będą brane pod uwagę przy aktualizacji feromonów (jeśli flaga nie została ustawiona przy aktualizacji będą brane pod uwagę wszystkie mrówki, które dotarły do końcowego wierzchołka).

W fazie aktualizacji feromonów wpierw następuje wyparowywanie feromonów, czyli zmniejszenie ich ilości na wszystkich krawędziach, a następnie mrówki zostawiają feromony na trasach, które przeszły.

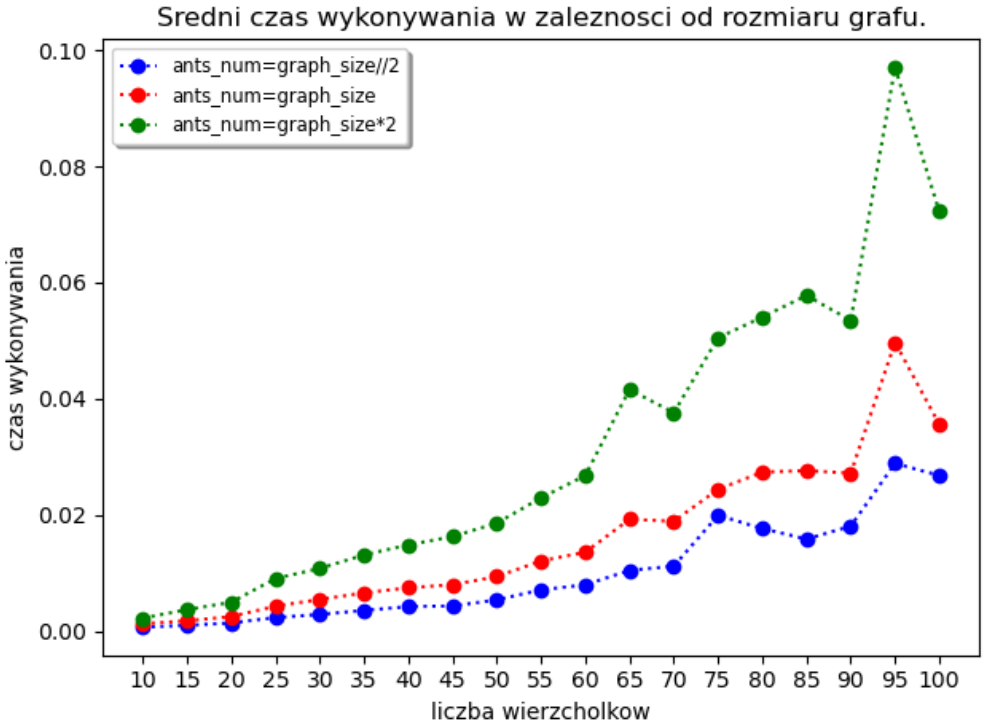
Algorytm zostanie zatrzymany w dwóch przypadkach – osiągnięta została maksymalna liczba iteracji lub przez zadaną liczbę iteracji kolejna znaleziona ścieżka jest nielepsza od poprzedniej.

Przeprowadzone testy:

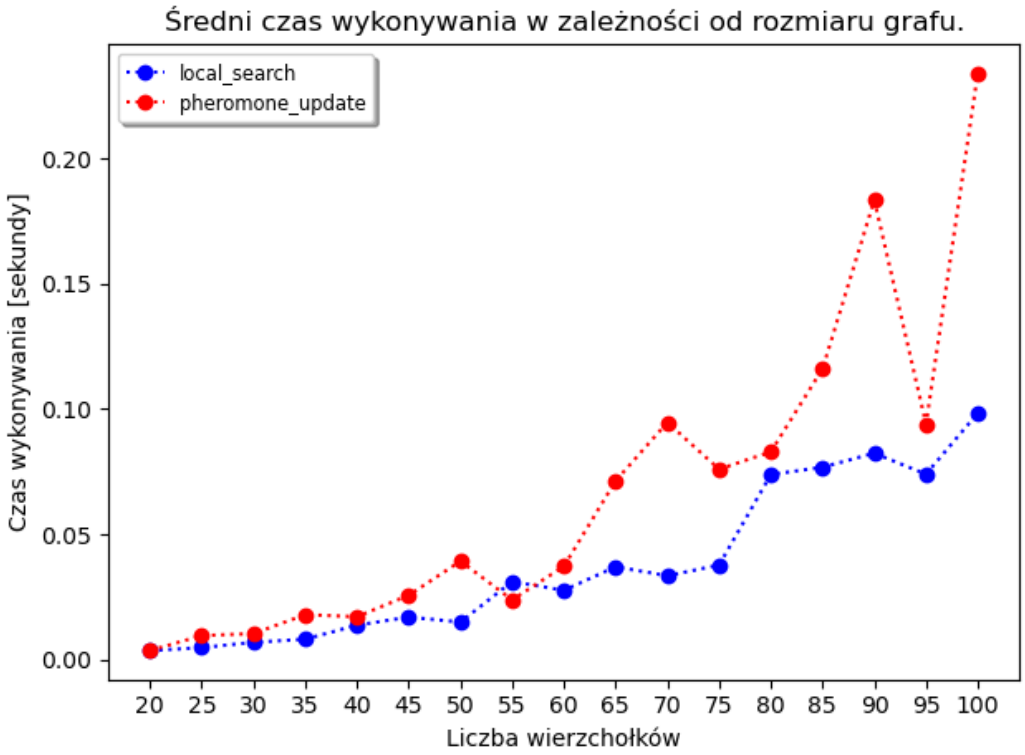
Przedstawione wyniki wydajności algorytmu ACO są efektem wielokrotnego wykonania algorytmu dla każdego badanego przypadku i uśrednienia wyników. Jednak duży udział losowości zarówno w generacji grafów do testów jak i tej zawartej w samym algorytmie ACO sprawia, że pozostawiają one wiele do życzenia, gdyż możliwości obliczeniowe dostępnego sprzętu nie pozwalały na dostatecznie wiele powtórzeń w racjonalnym czasie.
Działanie brute force zbadano jedynie dla stosunkowo niewielkich grafów, gdyż ze względu na jego złożoność obliczeniową jego działanie dla większych grafów było zbyt czasochłonne. Ze względu na brak losowości w działaniu tego algorytmu, nie powtarzano jego wykonania wielokrotnie dla tego samego wejścia.



Na pierwszym wykresie przedstawiono porównanie czasu wykonania się algorytmu ACO i brute force. Widać, że powyżej pewnej wielkości grafu algorytm mrówkowy staje się znacząco efektywniejszy ze względu na mniejszą złożoność.



Drugi wykres ukazuje zależność średniego czasu wykonania ACO od liczby wierzchołków w grafie i liczby mrówek w mrowisku. Zależność czasu od liczby wierzchołków jest w przybliżeniu liniowa, lecz wykazuje pewne charakterystyczne odchylenia dla większych grafów. Te odchylenia to cecha charakterystyczna tego algorytmu – ze względu na duży udział czynnika losowego, potrafi zachować się zupełnie nieprzewidywalnie i znaleźć rozwiązanie w znacznie dłuższym czasie niż spodziewany. Z wykresu można też zauważyć, że zwiększenie liczby mrówek nie wpływa dobrze na czas wykonywania się algorytmu – duża liczba obliczeń związanych z dodatkowymi mrówkami przeważa nad potencjalnie zmniejszoną liczbą iteracji potrzebnych do osiągnięcia warunku stopu .



Kolejny wykres ukazuje wpływ lokalnego przeszukiwania przestrzeni rozwiązań na zachowanie algorytmu ACO. Jak widać zastosowanie takiego kroku pozwala na częściową korekcję chaotycznych zachowań ACO, a niemal zawsze daje wynik tak samo szybko lub szybciej.

Algorytm mrówkowy ma jeszcze jedną istotną cechę – ze względu na to, że jest algorytmem probabilistycznym często daje rozwiązania suboptymalne, co w przypadku zastosowań, gdzie istotne jest maksymalne skrócenie ścieżki w grafie, będzie miało kluczowe znaczenie. Za to ACO może znaleźć dobre zastosowanie w problemach, gdzie koszt przejścia po krawędzi grafu zmienia się dynamicznie w trakcie poszukiwania optymalnej ścieżki, ze względu na elastyczność i zdolność dopasowywania się do zmian w grafie.

Instrukcja użytkownika
(dla Ubuntu 18.04.4 LTS)
1. Otworzyć terminal w głównym katalogu projektu
2. Uruchomić program poleceniem *python -m src.main [filepath]* gdzie filepath jest ścieżką do pliku zawierającego graf

Napisany interfejs pozwala też na dostrojenie parametrów algorytmu takich jak szybkość wyparowywania feromonów, maksymalna ilość iteracji etc. Aby skorzystać z tych możliwości należy uruchomić polecenie:
python -m src.main -h
lub
python -m src.main -- help