Zastosowanie algorytmu roju (PSO) do rozwiązywania problemu komiwojażera

Sprawozdanie 2 - Maciej Muszkowski, Maciej Roman

1. Algorytm ogólny

Do rozwiązania problemu komiwojażera zastosowaliśmy algorytm PSO (Patricle Swarm Optimalization – roju). Rój składa się z n cząsteczek, z których każda poszukuje najlepszego rozwiązania (w wypadku problemu komiwojażera jest to minimalizacja długości przebytej drogi). Kierunek przeszukiwania przestrzeni rozwiązań jest podyktowany zarówno poprzez indywidualne doświadczenie cząsteczki (najlepsze znalezione przez tą cząsteczkę rozwiązanie) jak i przez doświadczenie całego roju (najlepsze rozwiązanie znalezione przez rój). W ten sposób cząsteczki wzajemnie na siebie oddziaływują, unikają „utknięcia" w minimum lokalnym, stopniowo przenosząc się do coraz lepszych obszarów przestrzeni rozwiązań.

Zapis algorytmu w pseudokodzie:

```
init_population(); // losowe przypisanie położenia i prędkości
każdej cząsteczce
t=0;
do
      for i=1 to popSize do
            if(f(Xi(t)) < f(Pi)) Pi = Xi(t);
            Pg = min(Pall);
            updateVelocity(); // uaktualnienie prędkości cząstki
            updateSpeed(); // uaktualnienie położenia cząstki
      end for;
      t++;
until end_criterium_reached(); // póki nie osiągnięte kryterium
stopu
```

Przyjęte oznaczenia:

- t – iteracja.
- $f(X)$ – funkcja oceny (długość drogi).
- P_i – najlepsze rozwiązanie znalezione przez cząstkę i-tą. Wektor.
- P_g - najlepsze rozwiązanie (globalne, roju). Wektor.

- $X_i(t)$ – położenie i-tej cząstki w n-wymiarowej przestrzeni w iteracji t. Wektor.
- $V_i(t)$ – prędkość i-tej cząstki w n-wymiarowej przestrzeni w iteracji t. Wektor.

Kryterium stopu zostanie opisane w następnych rozdziałach.

Położenie i prędkość każdej cząstki, są uaktualniane w $updateVelocity$ oraz $updateSpeed$ na podstawie wzorów:

$$V_i(t+1) = \omega \cdot V_i(t) + C_1 \cdot (P_i - X_i(t)) + C_2 \cdot (P_g - X_i(t))$$
$$X_i(t+1) = X_i(t) + V_i(t+1) \quad (1)$$

Oryginalna wersja algorytmu operuje na liczbach zmiennoprzecinkowych, aby algorytm mógł rozwiązać problemu komiwojażera, niezbędne jest stworzenie jego wersji dyskretnej. Do implementacji został wybrany język C++.

2. Algorytm w wersji dyskretnej dla problemu komiwojażera

Dyskretna wersją algorytmu roju [1] działa na skończonym zbiorze stanów S, którym jest zbiór wszystkich cyklów Hamiltona. Pojedynczy cykl jest, więc pozycją cząstki w danej dyskretnej chwili czasowej.

Dodatkowo niezbędne jest określenie dyskretnej funkcji celu $f(x)$ działającej na pozycjach cząsteczek $x = (n_1, n_2, \dots, n_n, n_{n+1})$, gdzie $n_1 = n_{n+1}$. Funkcja oceny działa na skończonej liczbie stanów, oraz jej minimum jest w rzeczywistości rozwiązaniem optymalnym.

$$f(x) = \sum_{i=1}^{N-1} w_{n_i n_{i+1}} \quad (2)$$

Gdzie:

- i – Numer pozycji węzła w cyklu Hamiltona.
- $w_{n_i,n_{i+1}}$ – Waga krawędzi pomiędzy kolejnymi wierzchołkami w. Jeśli nie ma pomiędzy nimi krawędzi, to możemy stworzyć nową „wirtualną” krawędź o bardzo wysokiej wadze.

Ze względu na to, że położenie cząsteczki definiowane jest reprezentowane przez permutację składającą się z liczb całkowitych, niezbędne jest zdefiniowanie kilku dyskretnych pojęć i operacji zmieniających położenie takiej cząstki.

2.1.Prędkość

Prędkość jest operatorem działającym na pozycjach cząsteczek, zamienia ich obecną pozycję na inną. Mając N-elementową permutację, o długości $|v|$, składającej się z listy transpozycji możemy zdefiniować prędkość, jako:

$$v = \left((i_k, j_k)\right), i_k \in \{1, \dots, N\}, j_k \in \{1, \dots, N\}, k \uparrow_1^{|v|} \tag{3}$$

Co oznacza „zamień numer i_1 z j_1 , następnie numer i_2 z j_2, ..., na końcu zamień $i_{|v|}$ z $j_{|v|}$". Zerowa prędkość zdefiniowana jest jako $\emptyset$ czyli pusta lista. Warto zwrócić uwagę na to, że 2 takie listy mogą być w przybliżeniu równoważne np. $\left((1,3),(2,5)\right) \cong ((2,5),(1,3))$.

2.2.Przeciwieństwo prędkości

Przeciwieństwo prędkości zostanie wykorzystane w odejmowaniu.

$$\sim v = \left((i_k, j_k)\right), k \downarrow_1^{|v|} = \left(\left(i_{|v|-k+1}, j_{|v|-k+1}\right)\right), k \uparrow_1^{|v|} \tag{4}$$

Co oznacza „zrób takie same podstawienia, ale w odwrotnej kolejności". Negacja musi spełniać warunki $\sim\sim v = v$ oraz $v + \sim v = \emptyset$.

2.3.Suma pozycji i prędkości

Jeśli $P_2 = P_1 + v$ oraz:

$$\begin{cases} P_1 = (1,2,3,4,5,1) \\ v = ((1,2),(2,3)) \end{cases} \tag{5}$$

To podstawiając kolejno otrzymamy $(1,2,3,5,4,1)$ i ostatecznie $P_2 = (3,2,1,5,4,1)$.

2.4.Różnica dwóch pozycji

Rozważając dwie pozycje P_1 oraz P_2 definiujemy działanie dające w wyniku prędkość v obliczoną zgodnie ze wzorem $P_2 = P_1 + v$. Zgodnie z tą zasadą, jeśli $P_1 = P_2$to $v = \emptyset$.

2.5.Suma dwóch prędkości

Rozważając dwie prędkości v_1 oraz v_2 jako listy transpozycji, operację dodawania można zdefiniować jako połączenie dwóch list, eliminując jednocześnie powtórzenia. Zgodnie z tą zasadą $|v_1 + v_2| \leq |v_1| + |v_2|$.

2.6.Mnożenie prędkości przez liczbę

Niech c będzie liczbą zmiennoprzecinkową, a v modyfikowaną prędkością. Rozważając następujące przypadki możemy zdefiniować operacje mnożenia:

- Gdy $c = 0$ - wynikiem jest zbiór pusty.
- Gdy $c \epsilon (0,1)$ - wynikiem jest prędkość „obcięta" do zbioru $|cv|$, gdzie $|cv|$ jest największą liczbą naturalną mniejszą lub równą $c|v|$. Można zapisać to, jako:

$$cv = \left((i_k, j_k)\right), k \uparrow_1^{|cv|} \tag{6}$$

- Gdy $c \geq 1$ – oznacza to że mamy $c = k + c',\ k \epsilon N,\ c' \epsilon \langle 0,1 \rangle$. Tak więc można zdefiniować:

$$cv = \overbrace{v+v+\ldots+v}^{k\ razy} + c'v \tag{7}$$

- Gdy $c < 0$ – wynikiem jest $cv = -c(\sim v)$.

2.7.Aktualizacja pozycji i prędkości cząstki

Aktualizacja pozycji i prędkości cząstki zachodzi dokładnie tak samo jak w wersji ogólnej algorytmu, zgodnie ze wzorem 1, tylko, że z użyciem powyższych operatorów.

3. Założenia programu

3.1.Kryteria stopu

Wybrano 2 warunki, które prowadzą do zakończenia algorytmu, przy wyborze kierowano się ich niewielkim kosztem obliczeniowym:

- Kryterium 1: „Powolny rój" – należy porównywać prędkości wszystkich cząsteczek z ustaloną wartością progową. W przypadku, gdy cząsteczki w roju poruszają się poniżej dobranej eksperymentalnie wartości program kończy działanie.
- Kryterium 2: „Brak poprawy od długiego czasu" – jeżeli nie następuje poprawa wyniku od ustalonej eksperymentalnie liczby iteracji program kończy działanie.

3.2.Złożoność obliczeniowa

Złożoność obliczeniowa jest zależna od liczby cząsteczek roju oraz kryterium stopu, z tego powodu jest trudna do oszacowanie, gdyż liczba iteracji głównej pętli algorytmu może być zmienna. Jeśli przyjmiemy:

- i - liczba iteracji głównej pętli algorytmu.

- n - liczba cząsteczek.

To złożoność wyniesie $O(in)$, liczba iteracji jest zazwyczaj znacznie większa od liczby cząsteczek.

3.3.Wejście/wyjście programu

Program implementujący/testujący jest aplikacją konsolową. Dane wejściowe mogą zostać podane na 2 sposoby, wczytane z pliku lub wygenerowane losowo z podaną maksymalną odległością pomiędzy 2 punktami. Na wyjściu wypisana zostanie długość najkrótszego znalezionego cyklu, czas wykonywania obliczeń oraz liczba punktów wejściowych. Opcjonalnie możliwe jest wypisanie macierzy odległości pomiędzy punktami oraz porównanie wyniku z wynikami działania innych algorytmów.

3.4.Parametry

Dla algorytmu PSO możliwe jest ustalenie liczby cząsteczek roju i maksymalnej liczby iteracji nieznajdujących lepszego rozwiązania.

3.5.Struktury danych

Do reprezentacji grafu zawierającego odległości pomiędzy punktami, ze względu na to, że zakłada się, iż jest to graf pełny (możliwe jest połączenie dowolnego punktu z innym), wybrano reprezentację macierzową grafu.

3.6.Projekty testów

Program podlega dwóm kategoriom testów – jakościowych oraz wydajnościowych. Z tego powodu konieczne było zaimplementowanie innych algorytmów, z którymi porównywane są uzyskane wyniki. Są to:

- Algorytm losowego przeszukiwania przestrzeni rozwiązań.
- Algorytm symulowanego wyżarzania.

Porównywana jest długość najkrótszego znalezionego cyklu oraz czas takich poszukiwań dla różnych grafów.

4. Wykaz literatury

1. **Clerc, Maurice.** Discrete Particle Swarm Optimization. [Online] 2000. http://clerc.maurice.free.fr/pso/pso_tsp/Discrete_PSO_TSP.htm.