Algorytmy Metaheurystyczne 2023

Laboratorium 4-6 (na ocenę)

Termin oddania: 7 laboratorium

Heurystyki należą do grupy algorytmów, które nie gwarantują uzyskania rozwiązania optymalnego (w przeciwieństwie do algorytmów dokładnych), ani nie gwarantują jak duża będzie różnica względem optimum (w przeciwieństwie do algorytmów aproksymacyjnych). W niektórych przypadkach nie gwarantują nawet uzyskania rozwiązania dopuszczalnego. Z drugiej strony heurystyki są zwykle prostsze w użyciu i działają w krótszym czasie. Z tego względu niejednokrotnie heurystyki wykorzystywane są jako składniki większych i bardziej wyrafinowanych algorytmów.

Local Search Jest to klasyczna heurystyka, bazująca na założeniu, że decyzje optymalne lokalnie są również optymalne globalnie. Istotą jest to że każda decyzja (lokalna) podejmowana jest w sposób optymalny. Szybkość metody wynika z faktu, że decyzje podejmowane w każdym kroku są proste (tj. są to zwykle problemy obliczeniowe o bardzo niewielkiej przestrzeni rozwiązań, możliwej do efektywnego rozwiązania metodą przeglądu zupełnego).

Dla problemu komiwojażera (choć po niewielkich zmianach znajduje też zastosowanie dla zbliżonych problemów) w każdej iteracji algorytm stara się poprawić istniejące rozwiązanie i kończy się gdy nie udało się tego zrobić. Algorytm lokalnego poszukiwania wykorzystuje pojęcie otoczenia. Intuicyjnie otoczeniem (sąsiedztwem) $N(r)$ rozwiązania r nazywamy pewien zbiór rozwiązań „bliskich" r, które nazywamy sąsiadami r. Sama bliskość jest pojęciem względnym – otoczenie dla tego samego r zwykle można zdefiniować na różne sposoby (różne typy otoczeń). Rozmiar otoczenia to liczba sąsiadów. Uzupełnieniem jest pojęcie ruchu, które można opisać jako funkcję, której argumentem jest r, a wartością powstały sąsiad. Otoczenie r możemy więc alternatywnie zdefiniować jako zbiór rozwiązań możliwych do uzyskania za pomocą danego ruchu. Zwykle oczekuje się by otoczenie spełniało szereg własności:

1. $N(r)$ nie powinno zawierać r.
2. $N(r)$ nie powinno być zbyt duże ani puste.
3. Każdy element $N(r)$ powinien niewiele różnić się od r (ruchy powinny być elementarne).
4. Ruchy powinny być możliwie proste (niższy koszt konstrukcji i oceny sąsiada).
5. Dla dowolnych r_1 i r_2 powinien istnieć ciąg ruchów, którym można przejść od r_1 do r_2 (każde rozwiązanie powinno być osiągalne z każdego).

Przykładowym otoczeniem dla problemu komiwojażera (lub innych problemów reprezentujących rozwiązania za pomocą permutacji) jest otoczenie *invert*. Ruch dany jest funkcją *invert*(n, i, j), która polega na odwróceniu kolejności wierzchołków od i-tego do j-tego (jak duże jest otoczenie jednej permutacji dla n wierzchołków?). Dla przykładowego rozwiązania:

$$\pi = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$$

wykonanie ruchu *invert*$(\pi, 4, 7)$ da rozwiązanie pośrednie postaci:

$$\pi = (1, 2, 3, 7, 6, 5, 4, 8, 9, 10).$$

Algorytm opiera się na obserwacji, że ruchy typu *invert* prowadzą często do poprawy rozwiązania przez poprawienie („rozplątanie") charakterystycznych nieoptymalnych pętli, łatwych do zwizualizowania w euklidesowym wariancie problemu. Algorytm składa się z następujących kroków:

1. Wybierz rozwiązanie początkowe (w dowolny sposób), które staje się rozwiązaniem aktualnym.
2. Wyznacz wartość funkcji celu wszystkich sąsiadów rozwiązania aktualnego (dla otoczenia *invert*).
3. Jako kandydata do poprawy wybierz najlepszego z ocenionych sąsiadów.
4. Jeśli kandydat nie jest lepszy od aktualnego rozwiązania, to zakończ algorytm.
5. Zastąp aktualne rozwiązanie kandydatem i przejdź do kroku 2.

Zastanów się dlaczego ten algorytm dla grafów euklidesowych zawsze zwróci rozwiązanie bez krzyżujących się krawędzi. Znajdź prosty przykład, że nie jest to prawdą jeśli stosujemy zaokrąglenia takie jak w danych, które używamy.

Zadanie
Dla danych z poprzedniej listy oraz zbiorów `XIT1083`, `ICW1483`, `DJC1785`, `DCB2086` i `PDS2566` wykonaj następujące zadania:

1. Wylicz minimalne drzewo rozpinające i wykonaj $\lceil \sqrt{n} \rceil$ razy (n to liczba wierzchołków) następujący algorytm:
   (a) Wylosuj wierzchołek.
   (b) Skonstruuj cykl zaczynając przegląd drzewa od wylosowanego wierzchołka.
   (c) Zastosuj algorytm *Local Search* do tak uzyskanego cyklu.

   Dla każdych danych podaj wagę minimalnego drzewa rozpinającego, średnią wartość uzyskanego rozwiązania, średnią liczbę kroków poprawy oraz najlepsze uzyskane rozwiązanie.
2. Wykonaj algorytm *Local Search* dla n losowych permutacji. Dla każdych danych podaj średnią wartość uzyskanego rozwiązania, średnią liczbę kroków poprawy oraz najlepsze uzyskane rozwiązanie.
3. Aby przyspieszyć obliczenia, zamiast testować całe sąsiedztwo, wybieramy tylko najlepszego sąsiada z n losowo wybranych. Wykonaj tak zmodyfikowany algorytm *Local Search* dla n losowych permutacji. Dla każdych danych podaj średnią wartość uzyskanego rozwiązania, średnią liczbę kroków poprawy oraz najlepsze uzyskane rozwiązanie.

Przygotuj krótkie sprawozdanie z opisem uzyskanych wyników.

Kryteria oceny
Na 3.0 trzeba wykonać jedno z zadań, za każde kolejne dodaje się 1.0. Dodatkowe 0.5 za jakość sprawozdania (w tym odpowiedź na pytania zadane na liście).