Sprawozdanie:

Projektowanie Efektywnych Algorytmów

Dmytro Verkhovsky 231844

**Prowadzący:** dr inż. Dariusz Banasiak

# Zadanie projectowe numer 2

Algorytm Tabu search dla problemu komiwojażera

14.12.2017

## **Wstęp**

* 1. **Opis zadania projektowego**

Należało zaimplementować algorytm tabu search dla problemu optymalizacyjnego oraz wykonać pomiary czasu działania algorytmu w zależności od wielkości instancji oraz jakości dostarczanych rozwiązań a po tym rozwiązania porównywać z najlepszymi znanymi rozwiązaniami dla konkretnych danych testowych.

**1.2 Wstęp teoretyczny**

1.2.1 *Problem komiwojażera*

Problem komiwojażera jest zagadnieniem optymalizacyjnym, polegającym na znalezieniu minimalnego cyklu Hamiltona w pełnym graﬁe ważonym. Nazwa pochodzi od typowej ilustracji problemu, przedstawiającej go z punktu widzenia wędrownego sprzedawcy (komiwojażera): dane jest n miast, które komiwojażer ma odwiedzić, oraz odległość / cena podróży / czas podróży pomiędzy każdą parą miast. Celem jest znalezienie najkrótszej / najtańszej / najszybszej drogi łączącej wszystkie miasta, zaczynającej się i kończącej się w określonym punkcie.

1.2.2 *Algorytm tabu search*

Fred Glover w roku 1977 przedstawił pracę dotyczącą m.in. wykorzystania pamięci krótkotrwałej i długotrwałej w przeszukiwaniu lokalnym. Pamięć długotrwała miała służyć do zapamiętania najbardziej atrakcyjnych elementów przestrzeni przeszukiwań podczas, gdy pamięć krótkotrwała służyła do pamiętania ostatnich ruchów i miała ulegać nadpisywaniu w kolejnych iteracjach przeszukiwania. Wyżej wymienione pomysły stały się podstawą do działania Tabu Search i w roku 1986 Fred Glover wydał pierwszą pracę dotyczącą tej heurystyki. Co ciekawe Michael Hansen, również w 1986 roku wydał swoją pracę na temat podobnej heurystyki. Heurystyka o nazwie Steepest Ascent Mildest Descent Heuristic powstała niezależnie od pracy Freda Glovera.

Rozwój algorytmu poszukiwania TABU

**1986** *Stworzenie algorytmu Tabu Search*(TS) razem z idea metaheurystyki.

**1989** *Pierwsza implementacja algorytmu przeszukiwania tabu***.**

**1991** *Zastosowanie TS do harmonogramowania produkcji*według koncepcji Just-in-Time.

**1991** *Zastosowanie TS do planowania zadan produkcyjnych*z ograniczonym dostepem do narzedzi.

**1994** *Pierwsze zastosowanie metody tabu do układania tras**w procedurze*

*„Taburoute” opartej na metaheurystyce przeszukiwania tabu.*

**1999** *Zastosowanie TS do sekwencjonowania DNA*

**2001** *Unifikacja algorytmu TS*

**2003** *Ziarnista metoda TS*

**2004** *Łaczenie metaheurystyki TS z innymi metaheurystykami*w celu układania tras pojazdów

Przeszukiwanie tabu (TS) jest algorytmem metaheurystycznym, który wykonuje lokalne wyszukiwania, aby algorytm nie spadł w pułapkę przedwczesnego lokalnego optimum, które zmuszają jego do powrotu do wcześniejszych decyzji i pracy cyklicznej.

Tabu Search zaczyna się od oryginalnego rozwiązania. Każda iteracja generuje sąsiedztwo rozwiązań, a najlepsze z tych sąsiedst wybieramy jako nowe rozwiązanie. Niektóre atrybuty poprzednich decyzji są przechowywane w Tabu list, która jest aktualizowana na końcu każdej iteracji.

Wybór najlepszego rozwiązania w sąsiedztwie odbywa się w taki sposób, że nie akceptuje żadnego z zabronionych atrybutów. Najlepsze rozwiązanie jest obecnie aktualizowane, jeśli nowa, aktualna decyzja jest lepsza i wykonalna.

Procedura jest kontynuowana aż do spełnienia jednego z dwóch kryteriów zatrzymania, czyli maksymalnej liczby wykonanych iteracji i maksymalnej liczby iteracji, podczas których bieżąca decyzja nie ulegnie poprawie.

**1.3 Rozpatrywanie algorytmu**

W naszym projekcie robimy wyszukiwanie optymalnej scieżki przy pomocy zamiany dwóch sąsiednich miast miejscami między sobą i wybranie z tego najbardziej przybliżonego rozwiązania do wiadomego poprawnego rozwiązania z strony internetowej (<http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/>).

Zaczynamy nasz algorytm od wybrania losowego wieszchowka startowego. Robimy to za pomocą funkcji rand() , używając tej funkcji możemy otrzymać wszystkie wierzchołki z podanego przedziału, od zerowego do wierzchołka Size-1, gdzie Size to ilość miast.

Za tym rozpoczynamy szukanie ścieżki jaka ma najmniejszy koszt przechodzenia do następnych wieszchowków. Robimy to za pomocą dwóch funkcji if. W pierwszym sprawdzamy czy tę wierczhołek już był odwiedzony, a w drugim sprawdzamy czy nowe przejście jest mniejsze od poprzedniego. Zapisujemy tą śczieżke jaką optymalna i wyszukujemy długość tej śczieżki.

W funkcji głównej zamieniamy między sobą sąsiedni wierzchołki otrzymujemy nową długość i sprawdzamy czy ona jest mniejsza od poprzedniej minimalnej długośći.

**2. Plan eksperymentu**

- Czasy poszczególnych operacji mierzone są przy pomocy funkcji zawartych w bibliotece **<chrono>** będącej częścią standardu C++11.

- Będziemy też wczytywać już przygotowane dane z strony zamieszczonej w pliku.

-Test jaki mierzy czas dla rożnej ilości miast jest przeprowadzony 100 dla konkretnego rozmiaru miasta.

**3. Wyniki eksperymentu**

Wyniki czasowe dla rożnych ilości miast:

Czas dla wybranych plików gdzie ilość próbek jest równa 10, długość tabu listy jest też równya 10, a warunek krytyczny równy 5.

|  |  |  |
| --- | --- | --- |
| ilość miast | Error % | Czas, [s] |
| 17(gr.tsp) | 4,102 | 9,05E-05 |
| 17(br17.atsp) | 1,9844 | 8,29E-05 |
| 53(ft53.atsp) | 24,4318 | 2,42E-03 |
| 55(ftv55.atsp) | 29,19222 | 2,82E-03 |
| 280(a280.tsp) | 21,0716 | 0,515587 |
| 439(pr439.tsp) | 25,5213 | 2,915231 |

Ten wykres jest częścią poprzedniego wykresu, który obejmuje nie wszystkie miasta a tylko te które mają mniejszy czas, żeby skoki czasowe były widoczniejsze.

Czas dla wybranych plików gdzie ilość próbek jest równa 15, długość tabu listy jest równa 10, a warunek krytyczny równy 10.

|  |  |  |
| --- | --- | --- |
| ilość miast | Error % | Czas, [s] |
| 17(gr.tsp) | 5,1272 | 0,0001213 |
| 17(br17.atsp) | 3,86 | 2,11E-04 |
| 53(ft53.atsp) | 26,67 | 2,84E-03 |
| 55(ftv55.atsp) | 24,66778 | 3,65E-03 |
| 280(a280.tsp) | 20,86082 | 0,6886089 |
| 439(pr439.tsp) | 23,93877 | 3,348729 |

Ten wykres jest częścią poprzedniego wykresu, który obejmuje nie wszystkie miasta a tylko te które mają mniejszy czas, żeby skoki czasowe były widoczniejsze.

Czas dla wybranych plików gdzie ilość próbek jest równa 100, długość tabu listy jest równa 80, a warunek krytyczny równy 20.

|  |  |  |
| --- | --- | --- |
| ilość miast | Error % | Czas, [s] |
| 17(gr.tsp) | 2,82 | 0,0006864 |
| 17(br17.atsp) | 0,8957 | 0,0007077 |
| 53(ft53.atsp) | 24,5437 | 0,01809645 |
| 55(ftv55.atsp) | 29,3107778 | 0,01693384 |
| 280(a280.tsp) | 24,25221 | 2,059198 |
| 439(pr439.tsp) | 21,77078 | 8,683293 |

Ten wykres jest częścią poprzedniego wykresu, który obejmuje nie wszystkie miasta a tylko te które mają mniejszy czas, żeby skoki czasowe były widoczniejsze.

Czas dla wybranych plików gdzie ilość próbek jest równa 100, długość tabu listy jest równa 80, a warunek krytyczny równy 50.

|  |  |  |
| --- | --- | --- |
| ilość miast | Error % | Czas, [s] |
| 17(gr.tsp) | 4,6144 | 0,0006701 |
| 17(br17.atsp) | 1,8488 | 0,0007501 |
| 53(ft53.atsp) | 26,1318 | 0,01830032 |
| 55(ftv55.atsp) | 30,60393333 | 0,01657907 |
| 280(a280.tsp) | 23,16579 | 2,233724444 |
| 439(pr439.tsp) | 22,72562222 | 8,834814444 |

Ten wykres jest częścią poprzedniego wykresu, który obejmuje nie wszystkie miasta a tylko te które mają mniejszy czas, żeby skoki czasowe były widoczniejsze.

Czas dla wybranych plików gdzie ilość próbek jest równa 1000, długość tabu listy jest równa 650, a warunek krytyczny równy 100.

|  |  |  |
| --- | --- | --- |
| ilość miast | Error % | Czas, [s] |
| 17(gr.tsp) | 0,5128 | 0,0050897 |
| 17(br17.atsp) | 1,5394 | 0,0063824 |
| 53(ft53.atsp) | 21,5999 | 0,1463847 |
| 55(ftv55.atsp) | 27,53594444 | 0,160071 |
| 280(a280.tsp) | 21,09012222 | 18,1707 |
| 439(pr439.tsp) | 21,39765556 | 65,63195556 |

Ten wykres jest częścią poprzedniego wykresu, który obejmuje nie wszystkie miasta a tylko te które mają mniejszy czas, żeby skoki czasowe były widoczniejsze.

Czas dla wybranych plików gdzie ilość próbek jest równa 1000, długość tabu listy jest równa 650, a warunek krytyczny równy 200.

|  |  |  |
| --- | --- | --- |
| ilość miast | Error % | Czas, [s] |
| 17(gr.tsp) | 2,3074 | 0,0050004 |
| 17(br17.atsp) | 1,3118 | 0,0066733 |
| 53(ft53.atsp) | 25,0498 | 0,153478 |
| 55(ftv55.atsp) | 28,78733 | 0,1663325 |
| 280(a280.tsp) | 22,88629 | 19,49951 |
| 439(pr439.tsp) | 22,21581 | 68,0507 |

Ten wykres jest częścią poprzedniego wykresu, który obejmuje nie wszystkie miasta a tylko te które mają mniejszy czas, żeby skoki czasowe były widoczniejsze.

|  |  |  |  |
| --- | --- | --- | --- |
| Programowanie dynamiczne | | Tabu Search | |
| Ilość miast | Czas [s] | Czas [s] | Error % |
| 17(br17.atsp) | 0,110605 | 0,0050 | 2,915 |
| 17(gr17.tsp) | 0,208331 | 0,0066 | 1,311 |

Na powyższym wykresie możemy zobaczyć różnicę miedzy czasem działania algorytmu programowania dynamicznego i Tabu Search.

**4. Wnioski**

W tym projekcie zostały wykonane pomiary dla algorytmu Tabu Search dla problemu Komiwojazera .

Z powyższego wykresu możemy zobaczyć że algorytm Tabu Seacrh ma swoje wady i zalety.

Z jednej strony on jest w dwa razy szybszy nawet dla największej testowanej liczby próbek(1000), ale z innej strony on ma w sobie ukryty błąd , jaki możemy zobaczyć w powyższej tabeli i ten błąd jest mały dla takich plików(br17.atsp ma poprawną ściezkę o długość 39, błąd 2,9% oznacza że w Tabu Search średni wynik był 40), ale się zwiększa dla dużych plików. Jeżeli dobrze ustawimy parametry za pomocą których będzie działać algorytm: ilość iteracji, długość tabu listy i warunek krytyćny, to możemy uzyskać przybliżony wynik do poprawnego Jeszcze jedną zaletą jest to że działa dla dużych plików(np. pr439) dla których programowanie dynamiczne już nie działa bo potrzebuje dużą ilość pamięći i nawet jeżeliby była ta ilość pamięći to by trwało długo. Płacimy za tę szybki czas i małą ilość potrzebowanej pamięći dokładnością otrzymanych wyników.