Szymon Pilawka
Karol Krzekotowski

Określenie Problemu

Podobnie jak w poprzednim etapie wciąż zmagamy się z problemem komiwojażera, znalezienia rozwiązania bliskiego optymalnego możliwie niskim kosztem. W tym etapie tematem badań jest algorytm **Tabu Search**, w skrócie wyszukiwanie z listą zabronień (elementów zakazanych).

Metaheurystyka **Tabu** działa poprzez przeszukiwanie sąsiedztw dla obecnie rozważanej permutacji. Widać tu analogię do metody 2Opt, jednak w odróżnieniu od algorytmu optymalizującego, nie rozważamy całego sąsiedztwa, lecz jedynie jego fragment aby zwiększyć różnorodność rozwiązań. W naszej implementacji generujemy sąsiedztwo losowo (podejście probabilistyczne). Następnie przechodzimy dalej z nową najlepszą permutacją pod warunkiem, że nie znajduje się ona na liście zabronień.

Lista Tabu ma za zadanie zapobiegać nawrotom oraz zapobiegać utknięcia w pewnym minimum lokalnym. Jednak znalezienie się elementu na liście Tabu nie oznacza wykluczenia rozwiązania permanentnie (lista ma określoną długość i odrzuca najstarszy element). Może przyjmować różną postać, zakazywać ruchów bądź permutacji. W naszej implementacji na liście Tabu przechowujemy zabronione permutacje.

Badanie złożoności

Czas działania algorytmu jest w dużej mierze zależny od ilości wykonywanych iteracji. Przy nieograniczonej liczbie iteracji zawsze doszlibyśmy do najlepszego rozwiązania, jednak z punktu widzenia algorytmu nie mamy wiedzy że jest to minimum globalne (oraz lista Tabu nie jest nieskończona – przez co w końcu natrafimy na nawrot), przez co i tak należy nałożyć ograniczenie w celu zakończenia działania algorytmu.
Określmy **k** – liczba iteracji, **n** – rozmiar grafu, **t** – rozmiar listy tabu

Każda iteracja składa się z kroków: generowanie sąsiedztwa, porównywanie sąsiedztwa, wynik.
W implementacji sąsiedztwo jest liniowo zależne od rozmiaru grafu. Zatem krok generowania sąsiedztwa zajmuje nam O(n*op), gdzie op to koszt operacji invert/swap. Dla swap jest to wielkość stała, dla invert natomiast zależy od wylosowanego przedziału i może w najgorszym przypadku wynieść O(n).
Porównywanie sąsiedztwa rozumiemy poprzez wyznaczanie funkcji celu dla danej permutacji, następnie porównywanie z najlepszym dotychczas kandydatem z sąsiedztwa (czas stały).
Wyznaczenie funkcji celu ponownie zależy od metody, dla grafów symetrycznych zarówno wartości dla swap oraz invert możemy wyznaczyć w czasie stałym jeśli znamy wartość funkcji celu wejściowej permutacji. Natomiast dla asymetrycznego invert ponownie czas zależy od rozmiaru przedziału i w najgorszym przypadku może wynieść O(n).
Przed porównywaniem permutacji sprawdzamy czy nie znajduje się ona na liście Tabu. W implementacji lista przechowuje całe permutacje, zatem koszt wyszukania czy permutacja znajduje się na liście wynosi O(t*n).

Możemy zatem określić złożoność jednej iteracji jako:

$$O(n)+O(n)+O(tn^2) \; dla \; invert \; TSP \; i \; swap$$

$$O(n^2)+O(n^2)+O(tn^2) \; dla \; invert \; ATSP$$

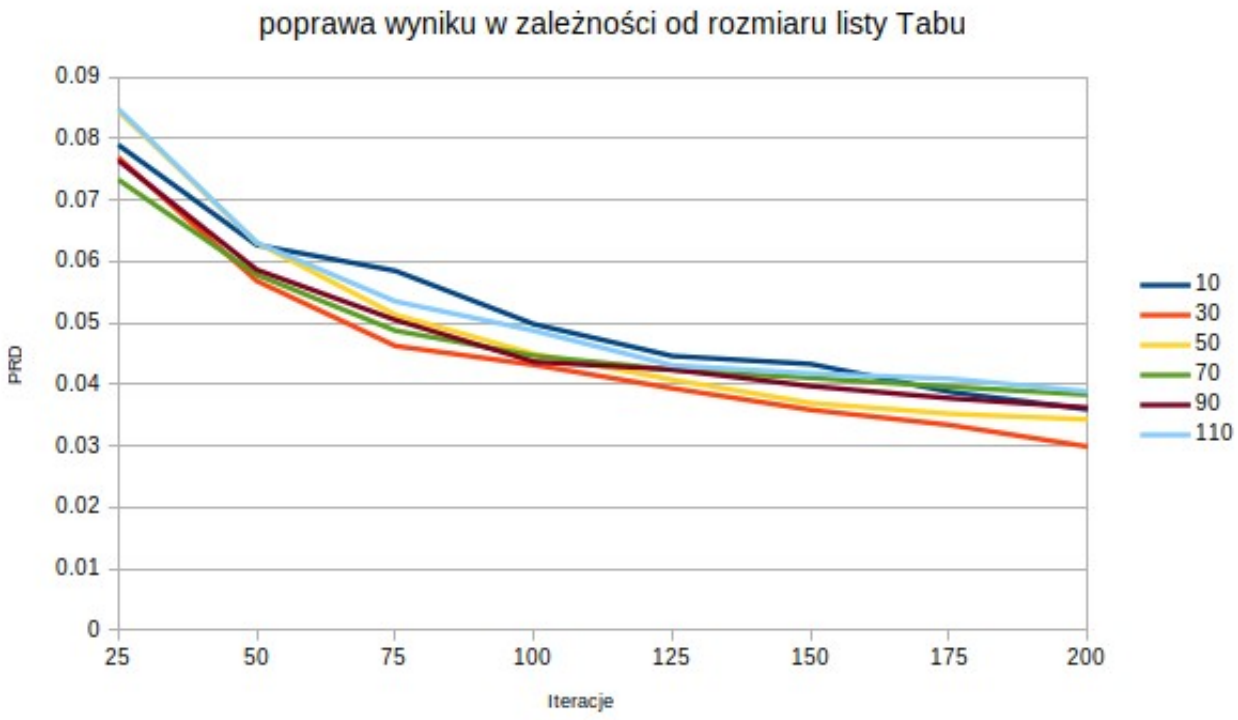
Zatem całkowita złożoność algorytmu wynosi:

$$O(ktn^2) dla\ wszystkich\ przypadków$$

Test – rozmiar listy Tabu

Lista Tabu jest istotnym elementem algorytmu, przechowuje ona elementy zabronione przy kolejnych iteracjach w celu zapobieganiu nawrotów oraz zwiększeniu różnorodności rozwiązań.
Jej długość ma również wpływ na złożoność czasową algorytmu, przy generowaniu sąsiedztwa sprawdzamy czy permutacja znajduje się na liście zabronień a co za tym idzie taki krok ma złożoność czasową równą długości listy.
Zbyt duża lista zabronień może spowodować utknięcie w pewnym punkcie. Możliwe jest, że permutacja dająca najlepsze perspektywy znajdzie się na liście, i przy długiej liście zabronień możemy nigdy już nie wrócić do takiego rozwiązania.
Zbyt krótka lista może spowodować zapętlenie w jakimś minimum lokalnym.

W teście porównuję rozmiary z zakresu 10 do 110, badam wpływ rozmiaru listy Tabu na poprawę wyniku w czasie. Używam 2Opt jako generatora rozwiązania początkowego z generowaniem sąsiedztwa poprzez invert. Uśredniam po 10 testów dla dwóch grafów TSP (rozmiar 107 i 150).



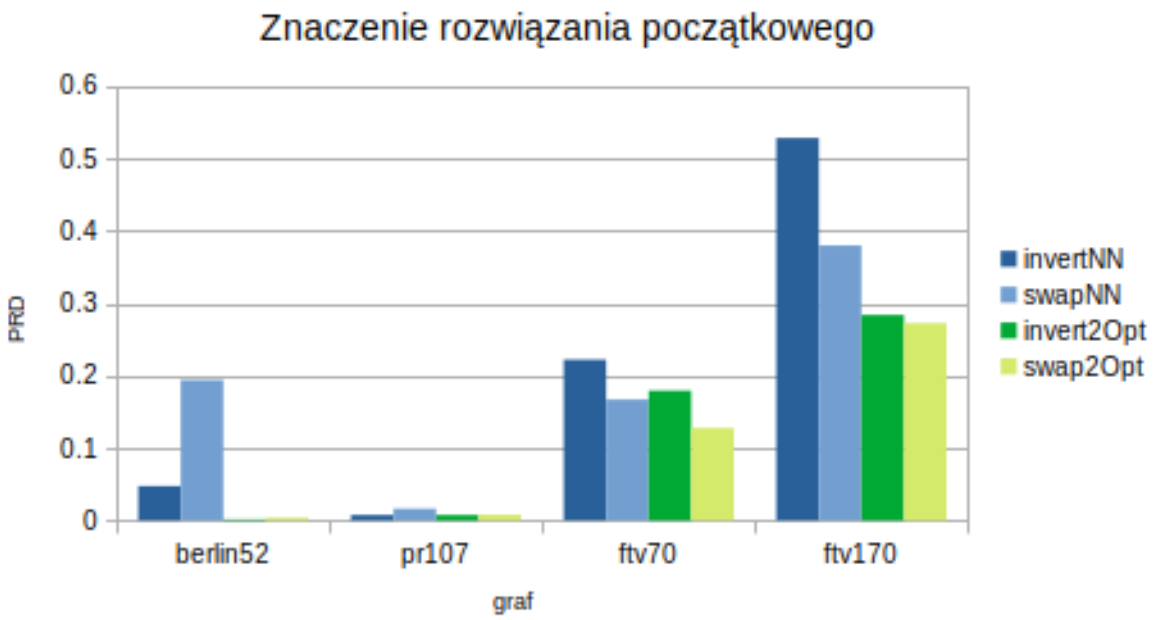
Najlepszą poprawę wyniku zauważamy dla rozmiaru rzędu 30. Zwiększenie rozmiaru ma niepożądany efekt blokowania optymalnych na dłuższą metę rozwiązań. Natomiast zbyt mała lista generuje nawroty w relatywnie krótkim czasie do reszty.

Test – wybór rozwiązania początkowego, porównanie SWAP vs INVERT

Wybranie permutacji początkowej ma znaczący wpływ na sposób w jaki algorytm poruszać się będzie po przestrzeni rozwiązań. Podanie bardziej optymalnej ścieżki na wejściu może ograniczyć przestrzeń po której będziemy się poruszać (od tego mamy listę Tabu – aby zwiększyć różnorodność). W teście porównywać będziemy dwie heurystyki – NN oraz 2Opt. Z poprzedniego etapu wiemy, że algorytm 2Opt znajduje lepsze rozwiązanie znacznie większym kosztem.

W drugiej fazie testu porównywać będziemy metody generowania sąsiedztwa. Tutaj na linię frontu idą dwie metody: invert oraz swap. Pierwsza z nich tworzy permutację poprzez zamianę kolejności wszystkich elementów w danym przedziale, natomiast swap jedynie podmienia dwa elementy.
O ile dla problemu symetrycznego operacje są znacząco małe, invert dla ATSP całkowicie zmienia wygląd ścieżki, co zbadamy dalej.

Test wykonywany był na 2 grafach TSP i 2 grafach ATSP. Po 10 powtórzeń dla każdej metody generowania sąsiedztwa i algorytmu początkowego. 1500 maksymalnych iteracji.

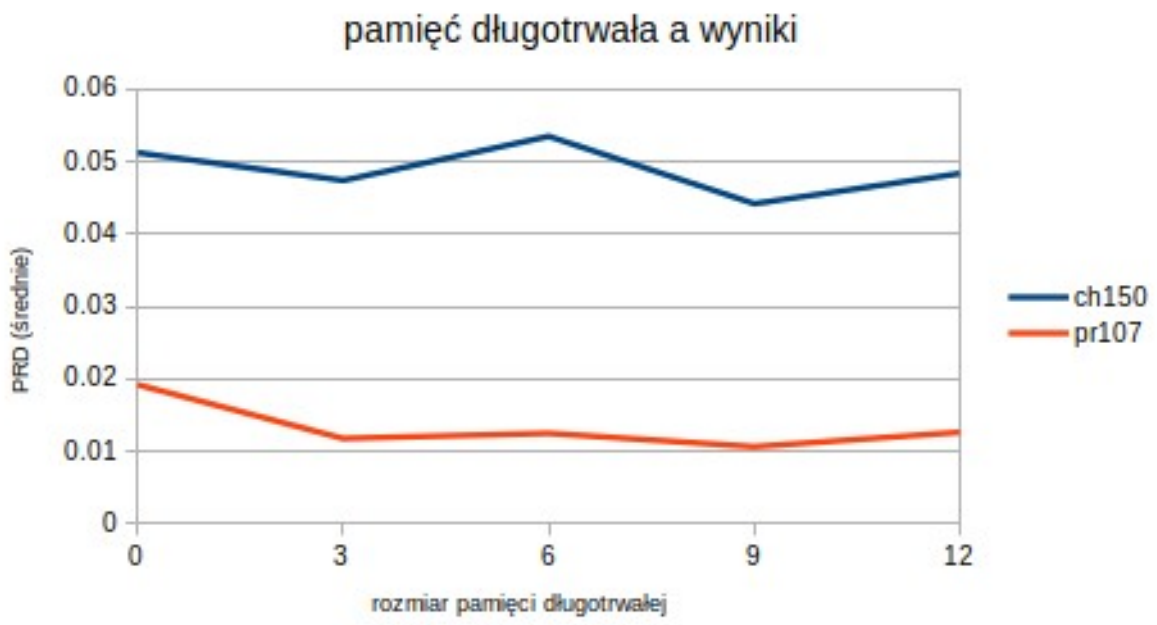


Rozpoczniemy od grafów symetrycznych (berlin52 i pr107). Tabu search z użyciem rozwiązania z 2Opt jako początkowego daje rozwiązania niemal optymalne dla danych grafów. Algorytm Tabu nie jest w stanie nadrobić przewagi jaką dawało przepuszczenie ścieżki przez 2Opt. Znacząco lepszą metodą generowania sąsiedztwa jest invert, przepięcie elementów jest bardziej skuteczne niż losowo zamienienie dwóch elementów na już zoptymalizowanej ścieżce.

Dla grafów asymetrycznych widzimy znaczącą różnicę. Lepsza metodą generowania sąsiedztwa okazuje się swap. Jest to spowodowane tym, że zmiany w ścieżce generowane przez invert są zdecydowanie zbyt duże. Szansa że trafimy na lepszą permutację są znikome.

Test – rozmiar pamięci długotrwałej oraz jej brak

Pamięć długotrwała jest bardzo użyteczna przy nawrotach i w dużej mierze definiuje możliwości algorytmy Tabu. Bez pamięci długotrwałej algorytm Tabu możemy przedstawić jako zmodyfikowana wersja algorytmu 2Opt – po prostu przeszukaj zbiór permutacji dla danego rozwiązania i znajdź minimum – utkniemy w pewnym minimum lokalnym i nie możemy wykonać nawrotu.
W teście jako rozwiązanie początkowe przyjąłem NN, działanie do 300 iteracji dla dwóch macierzy TSP po 10 powtórzeń.



Widzimy poprawę dla list długotrwałych rozmiarów długości różnej niż 6. Największą poprawę widać przy grafie pr107, gdzie PRD spada z 0.02 dla braku pamięci do prawie 0.01. Dla grafu ch150 możliwe że zauważylibyśmy podobny efekt dla większej liczby iteracji, gdyż brak pamięci długotrwałej oznacza utknięcie zawsze w tym samym dołku. W teście najlepiej wypada lista o długości 9.
Rozmiar pamięci długotrwałej możemy rozumieć jako szerokość spektrum w którym będziemy przeprowadzać poszukiwania. Dla małej liczby iteracji zbyt długa lista może oznaczać zbyt dalekie cofanie się przez co badamy rozwiązania mniej optymalne, jednak przy ogromnej ilości iteracji lepiej byłoby zwiększyć obszar poszukiwań.

Test – rozbieżność wyników jest podobna niezależnie od grafu (dla tej samej metody)

W teście statystycznie sprawdzam, czy różnice pomiędzy najlepszym a dowolnym rozwiązaniem są podobne niezależnie od badanego grafu dla tych samych metod i rodzajów grafów (z poprzedniego test wiemy, że symetryczność oraz metoda problemu dają różne wyniki). Badam metodę invert dla TSP, na dwóch grafach: gr120 i pr107.
Hipoteza: Dla obu tych grafów rozrzut wyników jest podobny.

Rozrzut rozwiązań wyliczam podobnie do PRD, jednak jako wartość referencyjną biorę minimum z otrzymanych rozwiązań a nie minimum globalne.

Wyniki test Wilcoxona porównujące rozrzuty par rozwiązań (100 par):

$$pvalue : 2.92 * 10^{-8}$$

Zatem jako że $pvalue < \alpha = 0.05$ odrzucamy hipotezę zerową i stwierdzamy z małym prawdopodobieństwem błędu, że rozrzut dla tych grafów nie jest podobny, a co za tym idzie w szerszym spektrum metoda nie gwarantuje podobnego rozrzutu dla każdej pary grafów.