Optymalizacja Kombinatoryczna – CVRPTW

Metoda – sparametryzowany algorytm losowy
Prowadzący – prof. Dr hab. inż. Maciej Drozdowski
Igor Simon – 155923
Bartłomiej Rudowicz – 155993
E-mail – igor.simon@student.put.poznan.pl
Data oddania sprawozdania:

1. Wstęp

CVRPTW jest problemem z klasy NP-trudnych. Ten problem zakłada, że każda dostawa musi być zrealizowana przez pojazd o ograniczonej pojemności, a każdy klient ma określony czas, w którym może być obsługiwany. Z racji, że problem jest wielokryterialny, celami kolejno są dopuszczalność rozwiązania, minimalizacja liczby ciężarówek, a następnie całkowity koszt.

Założenia są następujące:

a) Poruszanie się po liniach prostych między klientami;
b) Każdy klient odwiedzany jest tylko raz;
c) Dane wejściowe są w formacie Solomona;
d) Spełniane są ograniczenia dotyczące czasu oraz pojemności w wykonaniu obsługi klienta;
e) Wynik programu jest zwracany do pliku „*wynik.txt*";
f) Przy braku dopuszczalności rozwiązania program zwraca -1;
g) Przy dopuszczalnym rozwiązaniu program zwraca liczbę wykorzystanych ciężarówek, całkowity koszt trasy, a następnie każdą trasę, które są rozdzielane znakiem nowej linii.

2. Metoda rozwiązania problemu

Kryterium, według którego szukamy najlepszego rozwiązania jest minimalizacja wykorzystanej liczby ciężarówek, a następnie bierzemy pod uwagę jego koszt przy spełnieniu warunku równej liczby ciężarówek. Metodą, którą rozwiązujemy problem jest zmodyfikowany algorytm losowy. Najpierw sortujemy cały zbiór „bubble sortem" względem czasu końca okna dostawy, z którego są tworzone 2 podzbiory, jeden zawiera 4% elementów, którymi są klienci z największymi oknami czasowymi do zamknięcia, a drugi jest tworzony z pozostałych 96% elementów(wyjaśnienie, dlaczego taki podział został zastosowany znajduje się w punkcie *5.* sprawozdania). W tym przypadku jako kryterium parametryzujące stworzyliśmy procentowy podział podzbiorów. Staraliśmy się dobrać taki procent wielkości podzbiorów, aby wyniki były najbardziej korzystne. Większy podzbiór jest mieszany losowo za pomocą funkcji „shuffle" (aby poszerzyć zakres permutacji wykorzystaliśmy operator bitowy, całe wyjaśnienie tego problemu zostało umieszczone w bibliografii), który tworzy bazowo poprawne rozwiązanie **(jeśli nie ma klientów, którzy w żadnym wypadku nie będą w stanie zostać włączeni do rozwiązania)**, a następnie elementy z mniejszego zbioru staramy się włączyć do większego, sprawdzając wszystkie elementy w zbiorze po kolei (czy jest możliwość włączenia ich), w taki sposób aby wykorzystywać pozostałe luki pojemnościowe oraz czasowe (które spełniają wymagania), a tym samym poddawać zbiór minimalizacji wykorzystanych ciężarówek. Elementy, które nie mogły zostać wykorzystane, są dołączane na sam koniec jako nowa ścieżka.

Następnie proces mieszania podzbioru oraz dołączania do niego elementów jest powtarzany przez okres ograniczającego nas czasu.

3. Złożoność obliczeniowa

Aby móc rozpocząć analizę złożoności przyjmijmy, że n to liczba klientów, a k to liczba klientów w podziale, którym dołączamy elementy do ścieżek.
Wczytywanie z pliku ma złożoność O(n).
Aby przeanalizować złożoność obliczeniową naszej implementacji algorytmu rozbiliśmy całość na analizę 2 mniejszych problemów:

a) Sortowanie elementów-wykonuje się jednorazowo za pomocą sortowania bąbelkowego oraz ma złożoność $O(n^2)$, która wynika z liczby porównań oraz podwójnej pętli „for". Zdecydowaliśmy się na ten algorytm sortujący, gdyż dla struktury wektorów sortując względem jednej kolumny jest to problem łatwy w implementacji, jednak są efektywniejsze algorytmy sortujące (np. Quick Sort);
b) Pozostała część algorytmu która wykonuje się co każdą iterację:
    - Mieszanie elementów - wykorzystując wbudowaną funkcję „shuffle" ten etap ma złożoność O(n-k), gdyż k klientów zostało wyłączonych z mieszania;
    - Rozdzielenie problemu na ścieżki - wykorzystując jedynie jedną pętlę *for* oraz operacje porównujące osiągamy tutaj również złożoność O(n-k);
    - Dołączanie elementów do ścieżek - z racji, że próbujemy znaleźć pasujący element natychmiast do ścieżki, która z wymieszanej kolejności się zakończyła, to sprawdzamy wszystkie elementy z listy wstawiającej, zatem złożoność tego etapu wynosi O(k);
    - Dopisanie pozostałych elementów z podzbioru na koniec rozwiązania - z racji, że w najgorszym wypadku każdy element może być niewykorzystany, to złożoność tego etapu wynosi O(k).

Zapis do pliku ma złożoność O(n), gdyż w najgorszym przypadku będzie musiał wypisać n ścieżek (czyli każda ścieżka wówczas ma 1 klienta).

4. Szczegóły pomiarowe

Jako ograniczenie czasowe przyjęliśmy 5 minut, gdyż taka wartość była sugerowana w treści zadania, jednak w celach pomiarowych może ono zostać zmienione.
Z racji, że między punktami, dla których dokonano pomiaru, nie ma żadnych innych pomiarów, zastosowaliśmy wykresy punktowe bez linii.
Pomiary były dokonywane pod względami:

a) Rozwiązanie w czasie - dla czasów: 1 minuta, 3 minuty oraz 5 minut, pobieraliśmy aktualne najlepsze rozwiązanie, które było wykorzystywane do analizy;
b) Liczba klientów, która narastała o 100.

Same testy były przeprowadzane na danych z pliku *rc21010.txt*

5. Parametryzacja w celu poprawy rozwiązania

„Tuning" był dokonywany na danych z pliku *rc21010.txt* i algorytm był realizowany z ograniczeniem czasowym wynoszącym minutę o rozmiarach danych 100, 500 oraz 1000 czterokrotnie, by parametr był jak najbardziej uniwersalny.

Naszym parametrem, który ustawialiśmy w celu uzyskania najlepszego wyniku był podział tablic, z których jedną była lista do mieszania elementów, a druga do wstawiania. Ze względu na to, że naszym kryterium, według którego szukamy najlepszego rozwiązania jest najpierw liczba ciężarówek, a następnie, jeśli jest równa to koszt tras, tak samo wybieraliśmy, która konfiguracja parametru jest najlepsza. Poniżej zostały przedstawione wyniki, dla parametru procentowej wielkości listy wstawiającej elementy:

		Liczba uwzględnionych klientów			
	podział=5%	100	500	1000	
Próba badawcza w czasie 1 min	1	23	144	297	Liczba ciężarówek uzyskanych w rozwiązaniu
	2	24	143	292	
	3	23	145	293	
	4	23	144	295	

Rys 1.(Wyniki dla parametru podziału=5%)

		Liczba uwzględnionych klientów			
	podział=4%	100	500	1000	
Próba badawcza w czasie 1 min	1	23	130	272	Liczba ciężarówek uzyskanych w rozwiązaniu
	2	23	131	271	
	3	23	132	270	
	4	23	131	272	

Rys 2.(Wyniki dla parametru podziału=4%)

		Liczba uwzględnionych klientów			
	podział=3%	100	500	1000	
Próba badawcza w czasie 1 min	1	23	132	275	Liczba ciężarówek uzyskanych w rozwiązaniu
	2	26	132	275	
	3	26	132	273	
	4	26	132	273	

Rys 3.(Wyniki dla parametru podziału=3%)

		Liczba uwzględnionych klientów			
	podział=2%	100	500	1000	
Próba badawcza w czasie 1 min	1	23	143	302	Liczba ciężarówek uzyskanych w rozwiązaniu
	2	31	143	304	
	3	29	142	304	
	4	31	143	303	

Rys 4.(Wyniki dla parametru podziału=2%)

Jak można zauważyć, dla 4 prób badawczych w czasie wynoszącym 1 minutę najlepiej wypada parametr podziału o wartości 4% dla każdego przypadku, więc ten parametr został wybrany do przeprowadzenia dalszych pomiarów. Wartości w próbach badawczych mogą się nieznacznie różnić od siebie, ponieważ w dużej mierze nasz algorytm opiera się na losowości.

Również przeprowadziliśmy pojedyncze testy dla czasu wynoszącego 5 minut oraz liczbie wszystkich klientów w pliku (1000), aby potwierdzić, że parametr o wartości 4% jest najlepszym. Poniżej znajdują się wyniki dla każdego parametru:

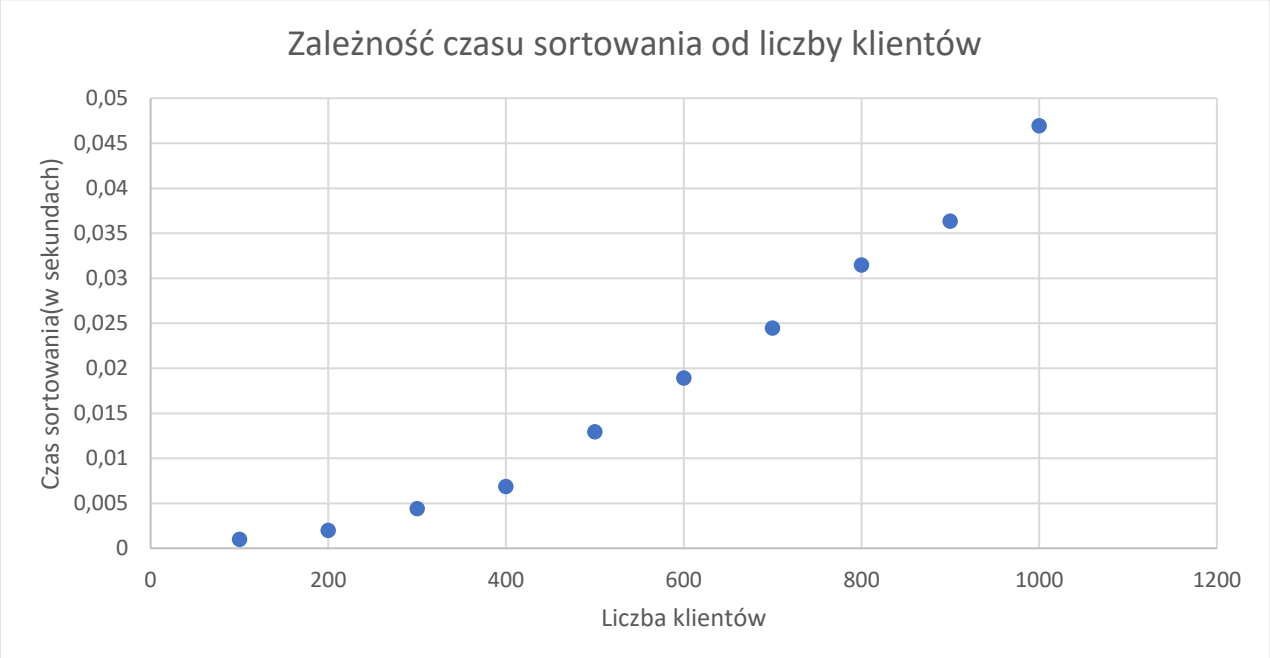
- 2%-302 ciężarówki;
- 3%-270 ciężarówek;
- 4%-268 ciężarówek;
- 5%-293 ciężarówki;

Jak widać, również w tym przypadku najlepsze wyniki są osiągane dla parametru wynoszącego 4% podziału.

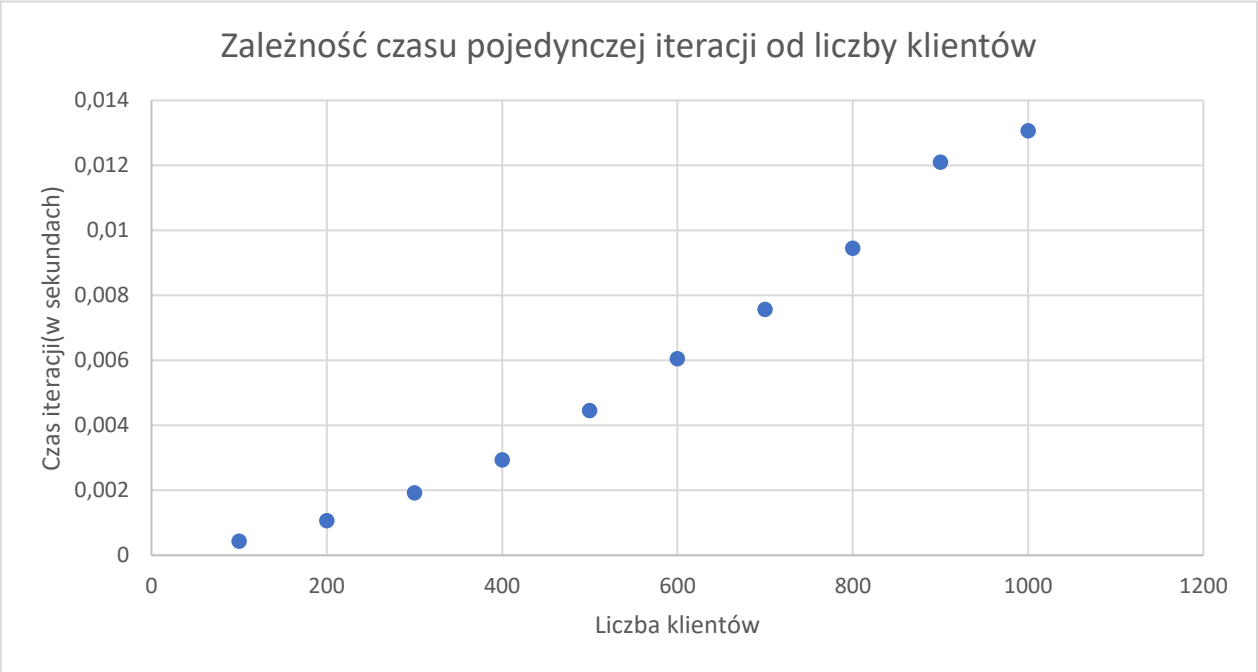
6. Ocena jakościowa oraz czasowa rozwiązania

W celach analizy czasu znajdowania rozwiązania, rozbiliśmy przedstawienie czasów na 2 wykresy. Pierwszy wykres oraz tabelka przedstawiają zależność czasu, w którym klienci są sortowani (gdyż klienci są sortowani jednorazowo) od liczby klientów, a drugi wykres i tabelka zależność czasu pojedynczej iteracji, gdzie elementy są dokładane do ścieżek od liczby klientów.

Liczba klientów	Czas sortowania[s]
100	0,000997
200	0,001994
300	0,004408
400	0,006879
500	0,012962
600	0,018935
700	0,024468
800	0,031499
900	0,036341
1000	0,046961



Liczba klientów	Czas pojedynczej iteracji
100	0,000437
200	0,001059
300	0,001925
400	0,002939
500	0,004455
600	0,006048
700	0,007564
800	0,009449
900	0,012097
1000	0,013067

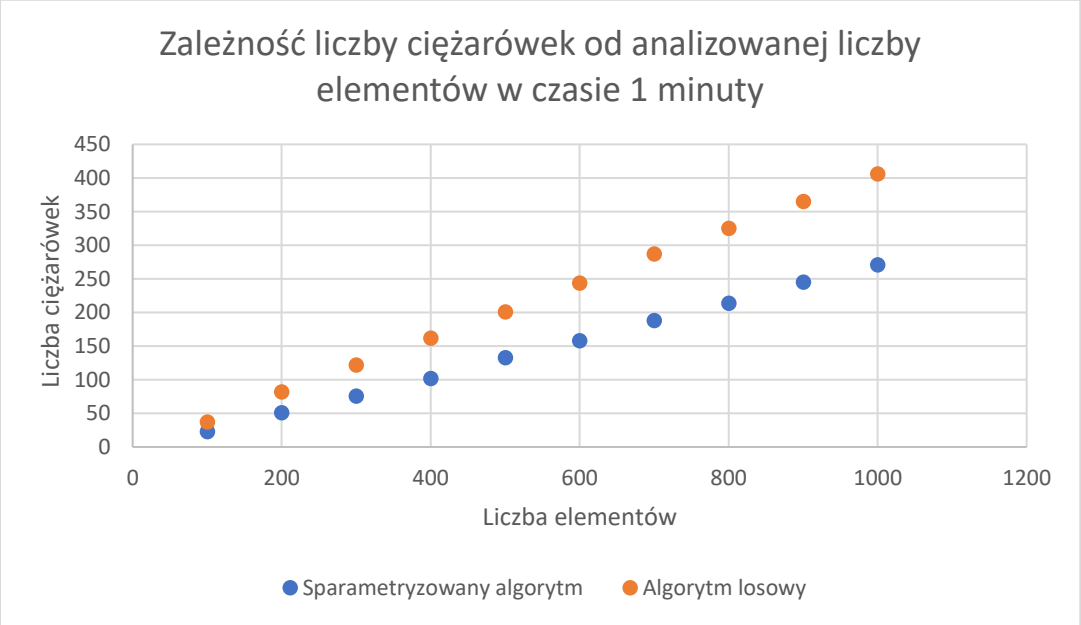


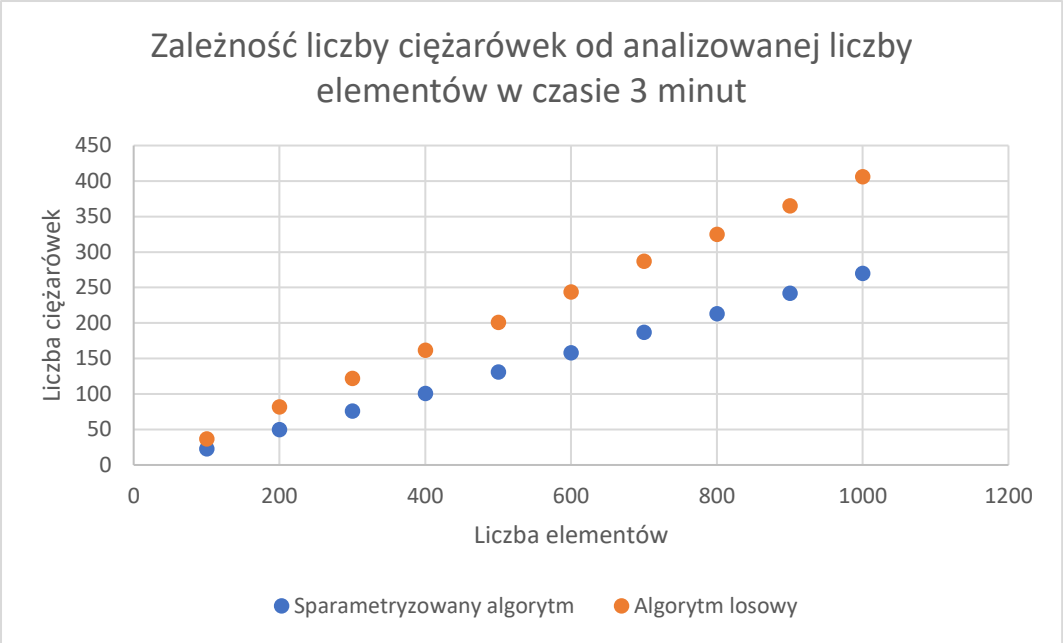
Jak można zauważyć, w przypadku sortowania oraz iteracji, czas przeznaczony na te dwie operacje wzrasta wraz z liczbą elementów, więc liczba klientów ma wpływ na czas wykonania algorytmu.

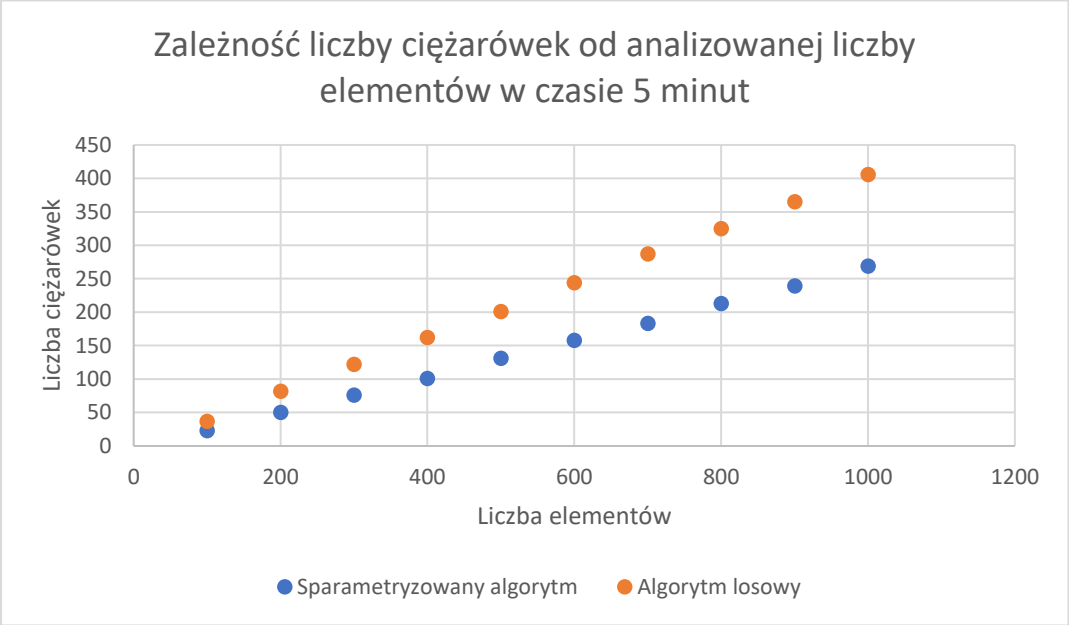
Aby dokonać analizy jakościowej naszego programu, postanowiliśmy porównać wyniki do podstawowego algorytmu losowego bez żadnych modyfikacji. Dokonywaliśmy porównań dla naszego algorytmu wykonywanego w czasach kolejno: 1 minuta, 3 minuty oraz 5 minut.

	Liczba ciężarówek			
	Sparametryzowany algorytm wykonywany w pewnym czasie			Algorytm losowy
Liczba elementów	1 minuta	3 minuty	5 minut	
100	23	23	23	37
200	51	50	50	82
300	76	76	76	122
400	102	101	101	162
500	133	131	131	201
600	158	158	158	244
700	188	187	183	287
800	214	213	213	325
900	245	242	239	365
1000	271	270	269	406

Rys 5. (Tabelka z liczbą ciężarówek)







Jak można zauważyć, liczba ciężarówek dla naszego algorytmu jest zdecydowanie mniejsza niż dla bazowego algorytmu losowego. Najczęściej rozwiązanie z upływem czasu jest poprawiane (liczba ciężarówek jest minimalizowana), jednak nie zawsze jest, ponieważ nie udaje się znaleźć w ograniczonym czasie zawsze lepszego rozwiązania ze względu na dużą liczbę możliwych permutacji. Poprzez parametryzację liczba permutacji została zmniejszona, jednak wciąż jest duża.

Również zdecydowaliśmy się na porównanie jak koszt trasy zmienia się względem liczby elementów.

	Koszt trasy			
	Sparametryzowany algorytm wykonywany w pewnym czasie			Algorytm losowy
Liczba elementów	1 minuta	3 minuty	5 minut	
100	109065.36041618	109065.36041618	109065.36041618	118416.03286920
200	254846.75791441	249301.07492181	249301.07492181	282220.96131909
300	371579.71220116	371579.71220116	371579.71220116	418385.13628821
400	488279.25060515	488075.10309376	488075.10309376	555998.64390788
500	615277.89264012	611390.99364990	611390.99364990	680050.14059103
600	725099.29244135	725099.29244135	725099.29244135	817394.22020397
700	844266.99827799	841261.59153990	835663.13974476	940692.53583160
800	964895.06789853	966219.05125326	966219.05125326	1082983.10285193
900	1076795.52205735	1073082.00854596	1075684.89204564	1189715.28967726
1000	1191621.35084952	1189976.09016723	1191908.42197790	1320399.69637135

Rys 6.(Tabelka z kosztami trasy)

W kosztach trasy nasz algorytm osiąga lepsze wyniki mając mniejsze koszty trasy od algorytmu losowego. Jednak porównując Rysunek 5. i Rysunek 6. można zauważyć, że przy minimalizacji liczby ciężarówek może zdarzyć się przypadek, że koszt trasy wzrasta.

7. Wnioski

Z przeprowadzonych testów można wyciągnąć następujące wnioski:

- Ta metoda jest szybka i daje dopuszczalne rozwiązanie, lepsze od typowego algorytmu losowego, jednak nie są to najczęściej optymalne wyniki;
- Losowość (przez parametr już ograniczona) może wpływać na analizę szerokiej puli rozwiązań, jednak to nie gwarantuje poprawy jakości rozwiązania w czasie;
- Można zauważyć związek między czasem działania algorytmu i jakością rozwiązań, jednak jest on nieznaczny;
- Przez priorytet na minimalizację liczby ciężarówek, czasem koszt trasy może wzrosnąć.

8. Bibliografia

https://www.cs.put.poznan.pl/mdrozdowski/dyd/ok/index.html#info
https://stackoverflow.com/questions/32586825/why-is-stdshuffle-as-slow-or-even-slower-than-stdsort
https://www.cs.put.poznan.pl/mhapke/TO-Ant.pdf
https://stackoverflow.com/questions/39288595/why-not-just-use-random-device
https://neo.lcc.uma.es/vrp/