Maksymilian Piotrowski & Jakub Grzebielec

Algorytmy Metaheurstyczne – Lista 1 – Sprawozdanie

Link do repozytorium: https://github.com/Mkanobi/Metaheurystyka-Piotrowski-Grzebielec

Zajęcia 2:

Zaimplementowano możliwość wczytywania, oraz generowania problemów TSP w formie:

- Euklidesowej przestrzeni z węzłami jako punktami w przestrzeni.

- TSP z podanymi odległościami między węzłami.

- ATSP z podanymi odległościami między węzłami.

Funkcje umożliwiające generację znajdują się w ./problem_render.py. Wczytywanie przeprowadzamy za pomocą biblioteki tsplib95.

Zaimplementowano także funkcję celu liczącą długość podanego cyklu Hamiltona, znajduje się ona w ./goal_function.py

Zajęcia 3:

Zaimplementowano algorytmy: k-random, nearest neighbour, extended nearest neighbor oraz 2-OPT. Znajdują się one w ./algorithms.py.

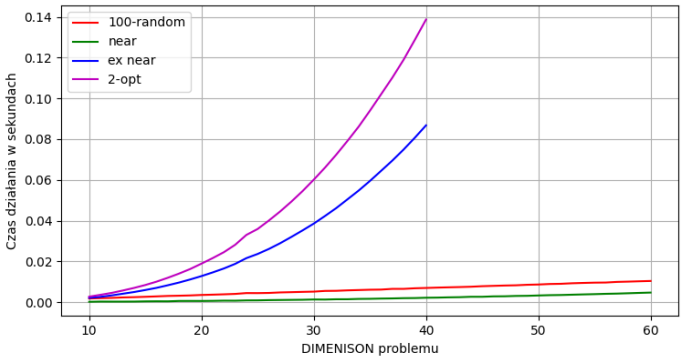
Algorytmy te można przetestować na danych z pliku, lub nowo wygenerowanych z dowolnymi parametrami przy pomocy ./run_algorithms.py uruchomionym z odpowiednimi argumentami.

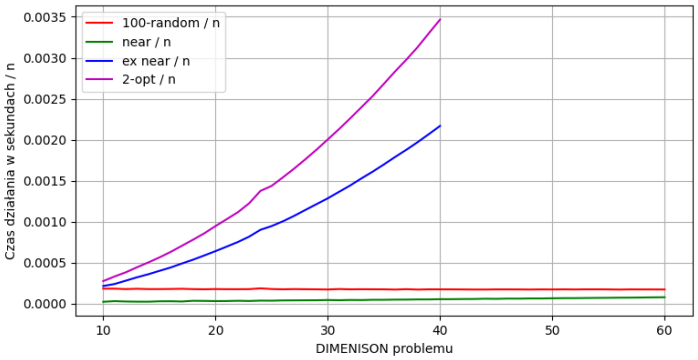
Zajęcia 4:

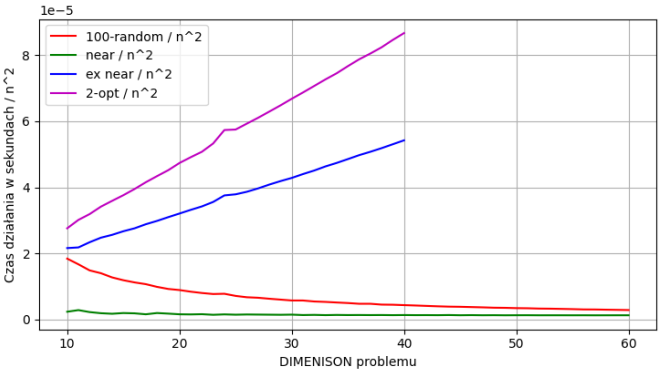
Przeprowadzono pomiary na utworzonych algorytmach i przeanalizowano wyniki.

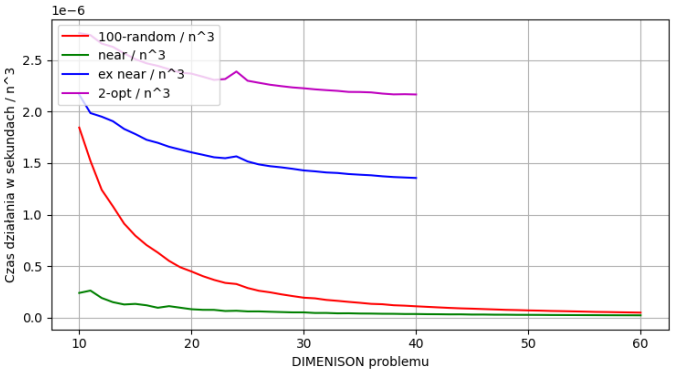
Analiza złożoności czasowej algorytmów:

Średni czas działania algorytmów w zależności od DIMENSION (z 100 generowanych problemów ATSP_EXPLICIT dla każdego DIMENSION)

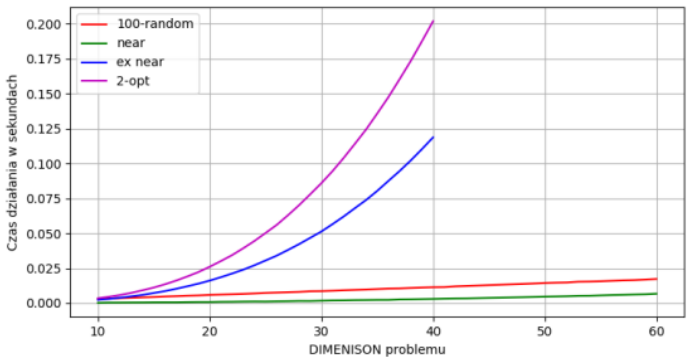


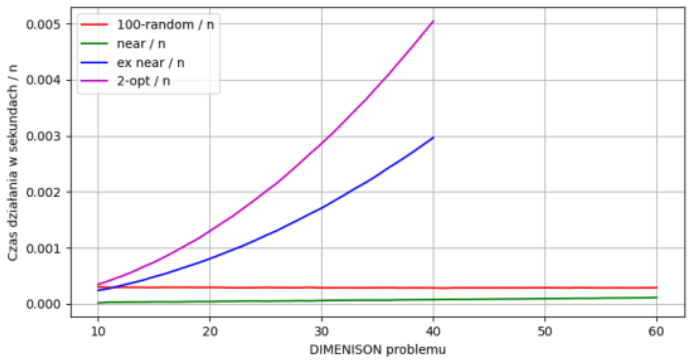


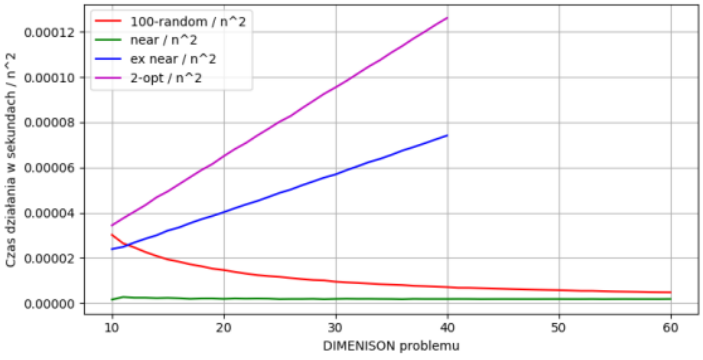


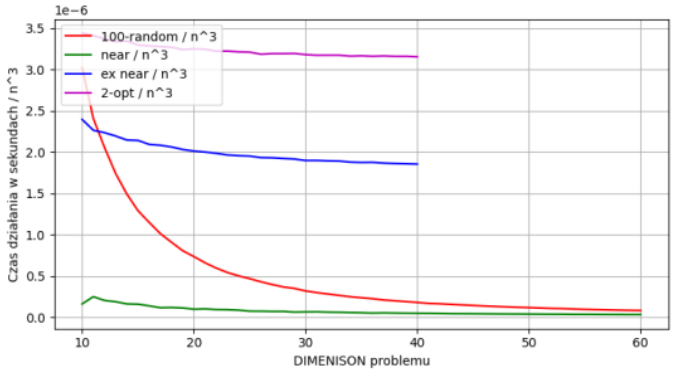


Średni czas działania algorytmów w zależności od DIMENSION (z 100 generowanych problemów EUC_2D dla każdego DIMENSION)

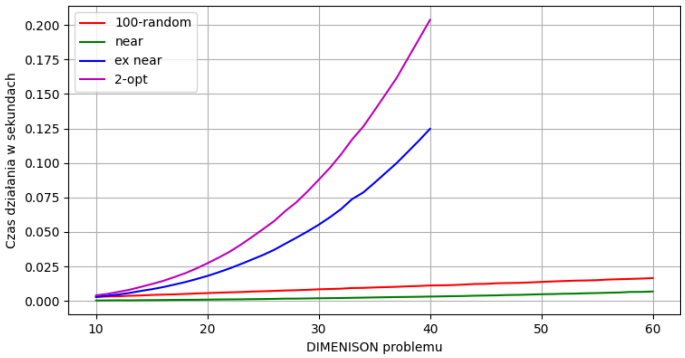


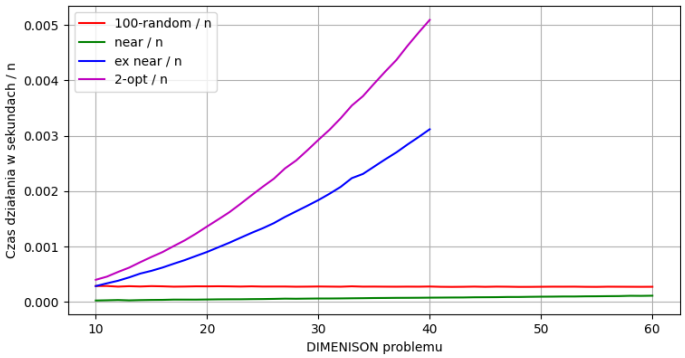


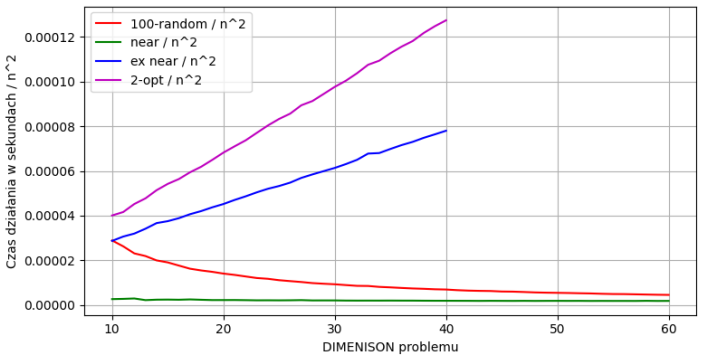


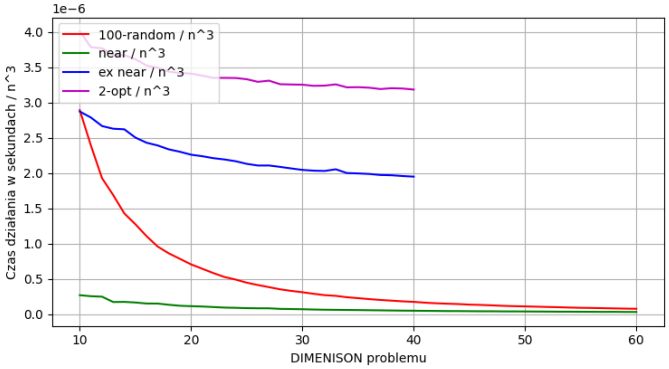


Średni czas działania algorytmów w zależności od DIMENSION (z 100 generowanych problemów TSP_EXPLICIT dla każdego DIMENSION)

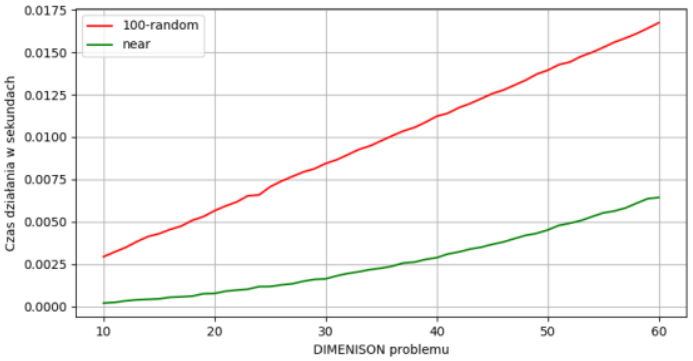


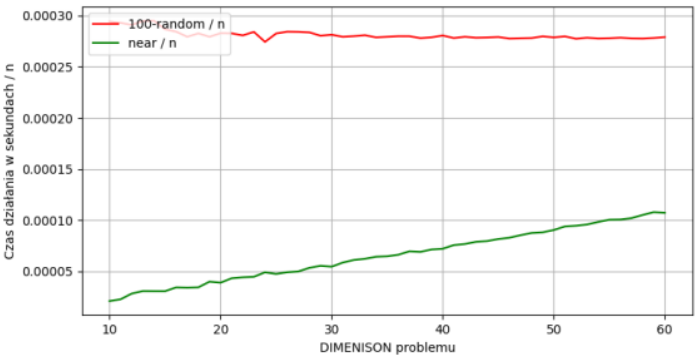


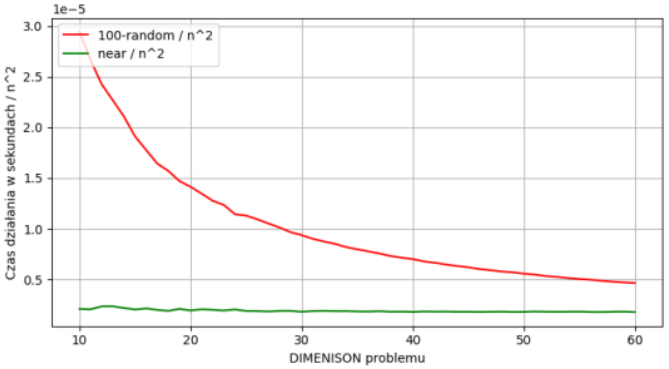


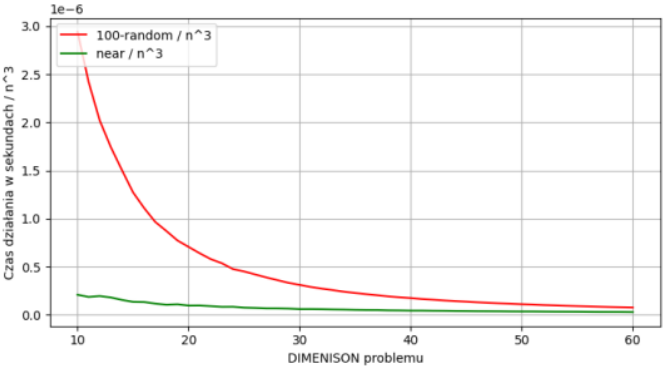


Średni czas działania algorytmów w zależności od DIMENSION (z 100 generowanych problemów EUC_2D dla każdego DIMENSION)









Jak widać dla wszystkich czterech wygenerowanych wykresów wyniki są bardzo podobne.

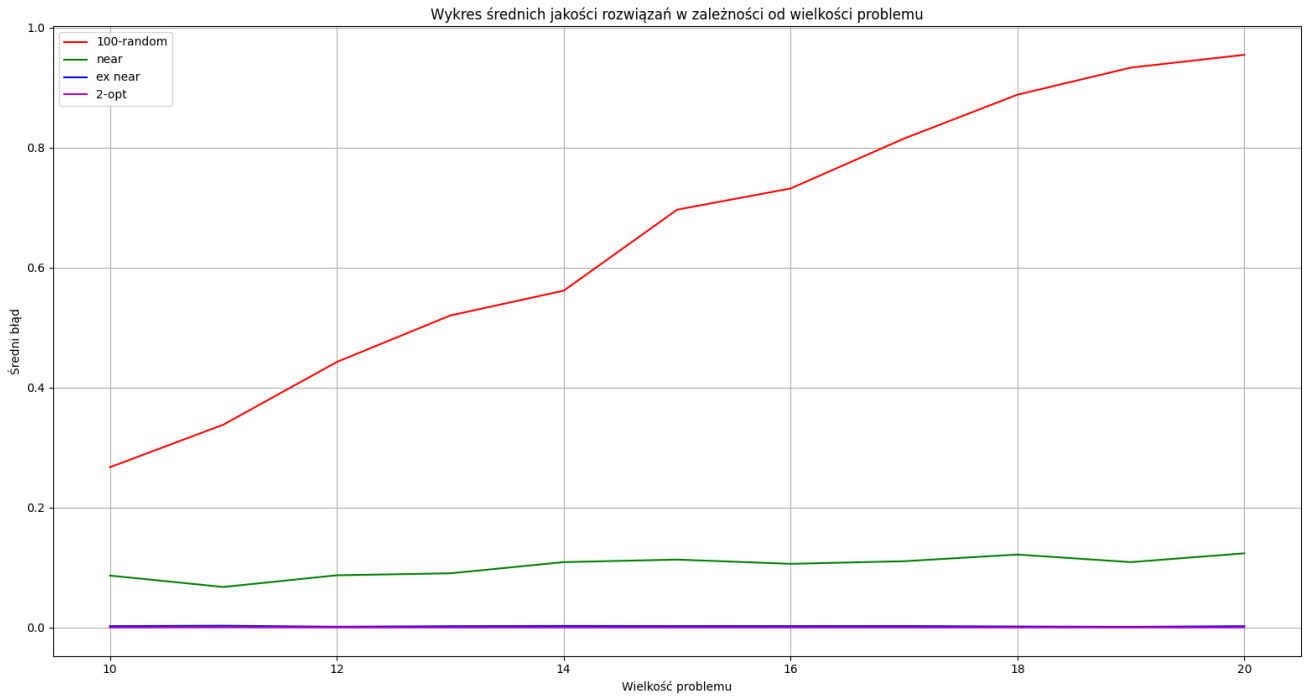
K-random polega na kilkukrotnym losowoaniu n-długiej permutacji i wyliczeniu dla nich funkcji celu. Obie te operacje są O(n). Zatem algorytm jest O(n). Potwierdza to wykres: „spłaszcza się” po podzieleniu przez n.

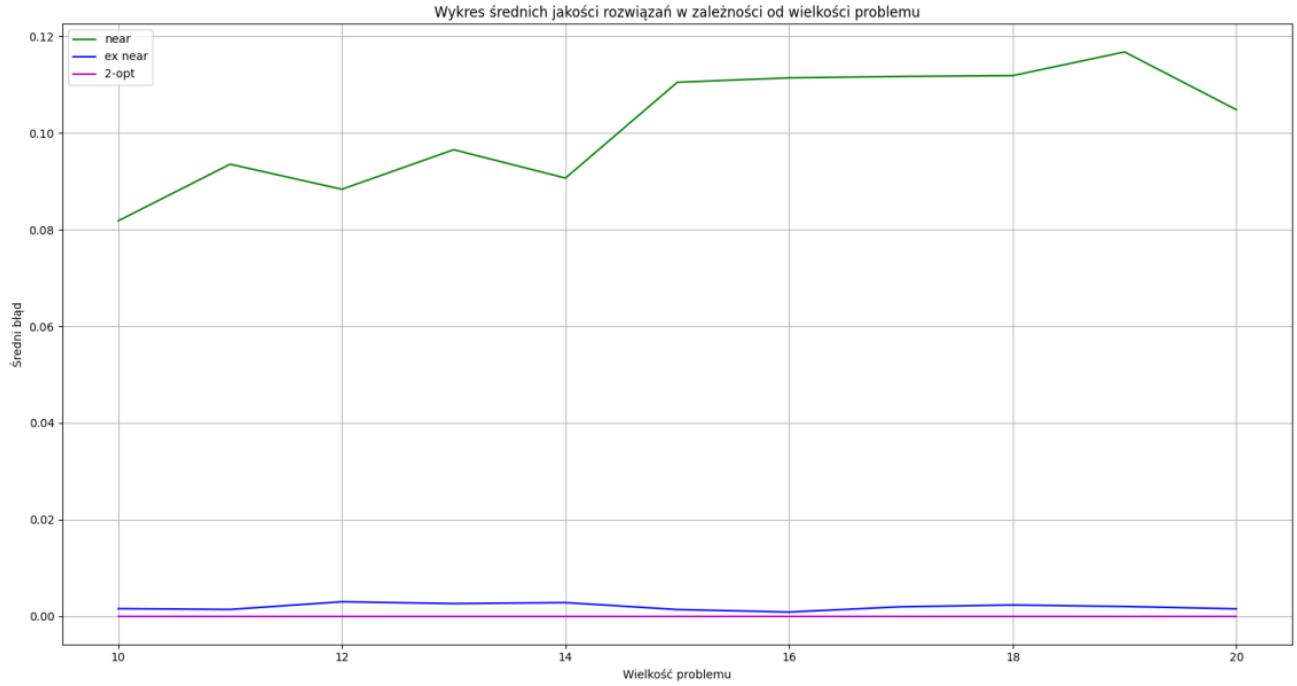
Średni czas działania Nearest Neighbour przestaje być rosnący przy podzieleniu średniego czasu przez romiar problemu do kwadratu (lewy dolny róg). Rzeczywiście NN ma złożoność O(n^2), bo dla każdego z n węzłów szuka wśród pozostałych najbliższego.

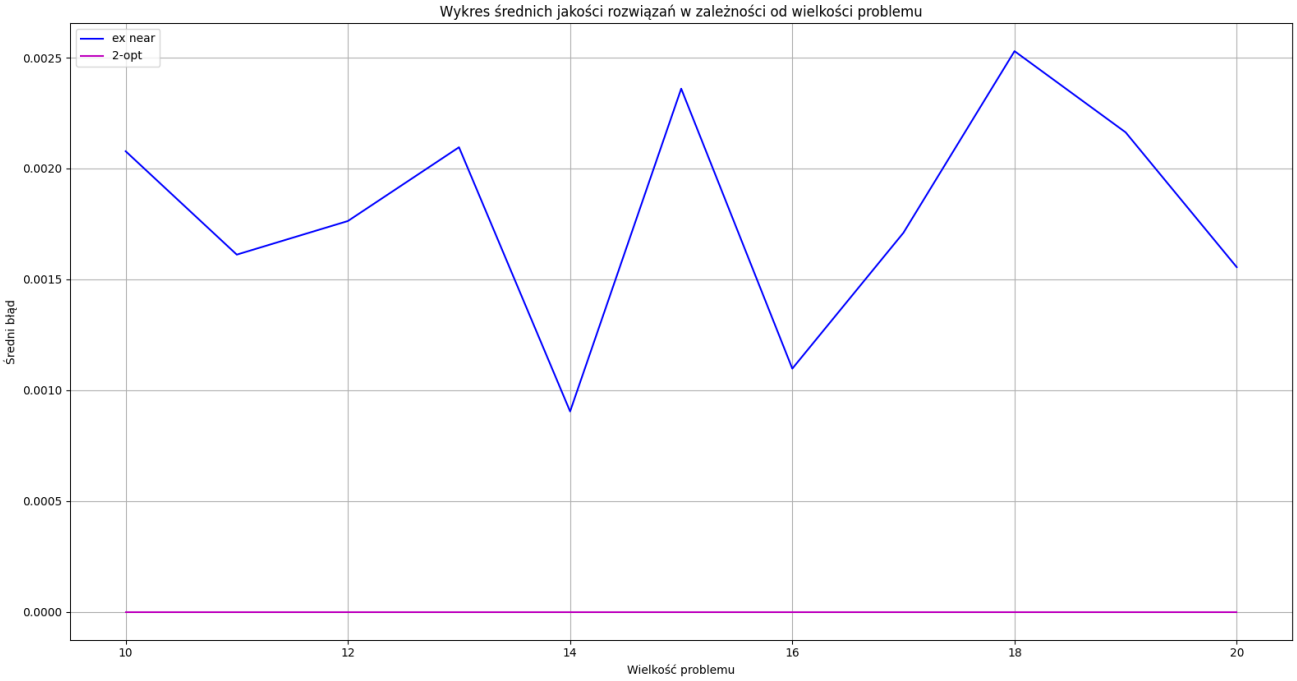
Czasy ENN przestają rosnąć wraz z wielkością problemu dopiero przy podzieleniu przez tą wielkość do potęgi 3. Także jest to zgodne z oczekiwaniami – ENN wykonuje NN n razy, czyli jest O(n^3).

Czas działania 2-OPT rośnie asymptotycznie tak samo szybko jak ENN, gdyż startowym rozwiązaniem 2-OPT’a jest właśnie ENN, a same operacje inverse zajmują widocznie także O(n^3).

Analiza PRD algorytmów:







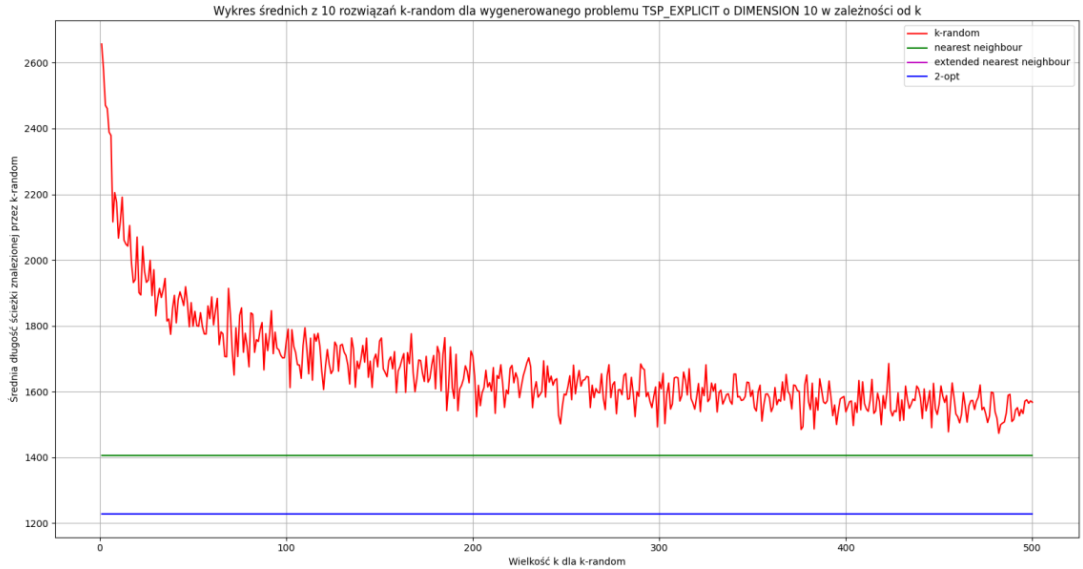
Zgodnie z przewidywaniami najniższe średnie PRD ma zawsze 2-OPT, którego startowym rozwiązaniem jest ENN.

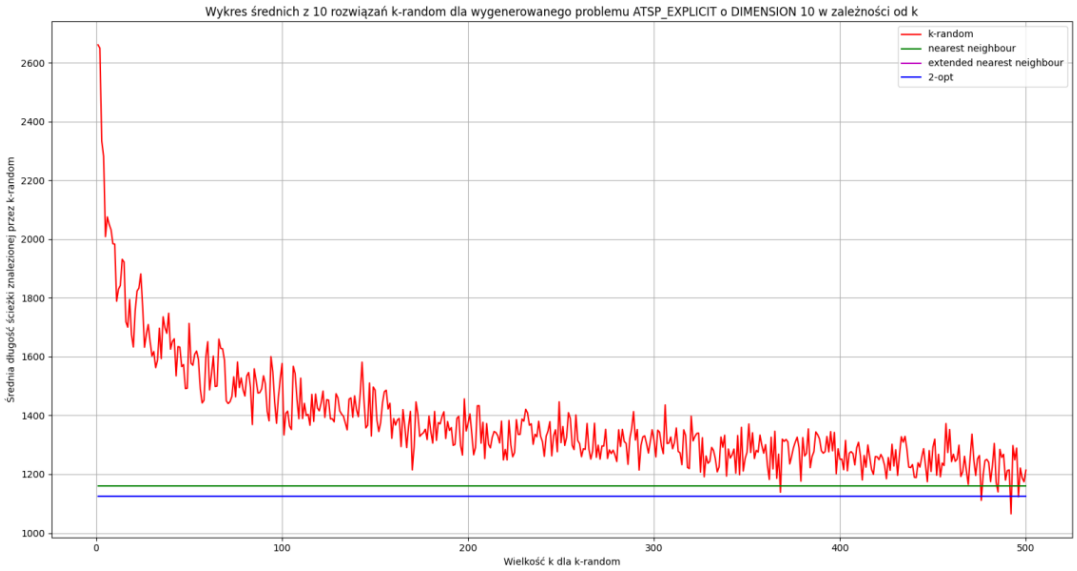
Drugie miejsce zajmuje ENN, którego średnie PRD jest niewiele mniejsze od 0, co pokazuje, że często rozwiązanie ENN nie daje się znacząco ulepszyć invertami w 2-OPT.

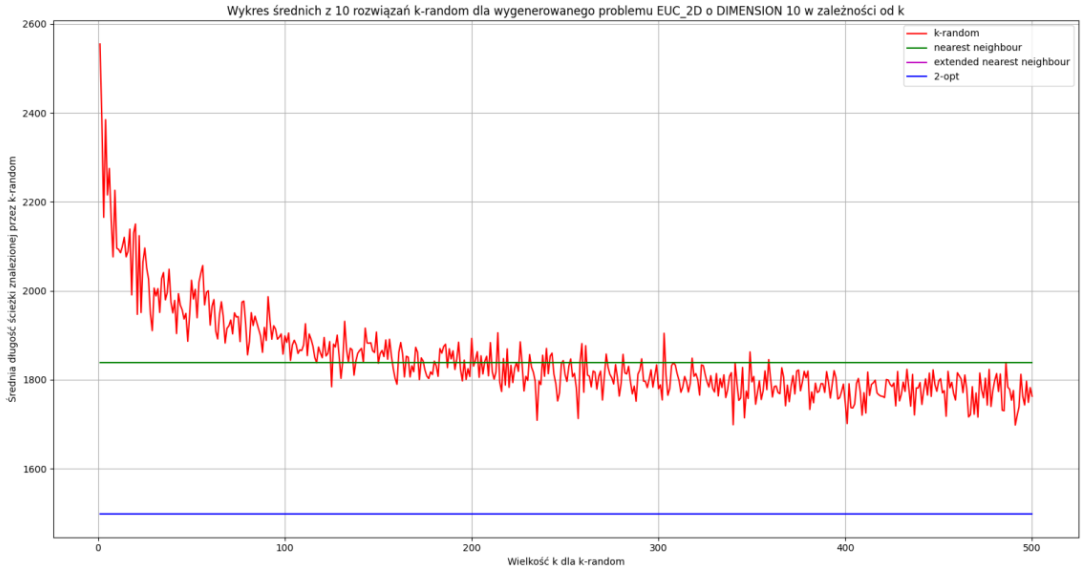
Oczywiście NN ma zawsze wyższe PRD niż ENN.

PRD k-random rośnie razem z rozmiarem problemu.

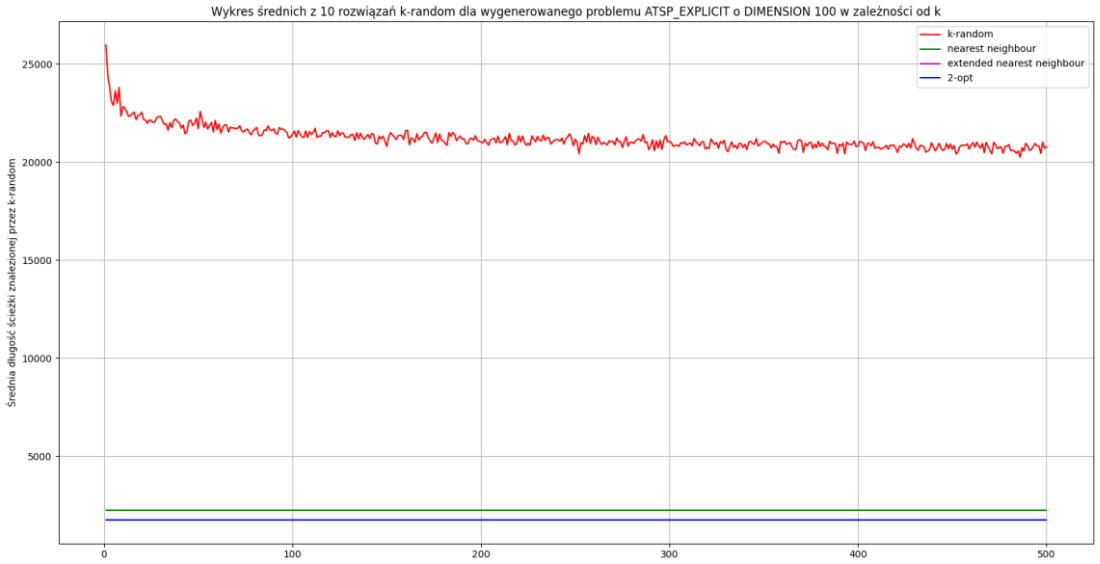
Analiza k-Random:

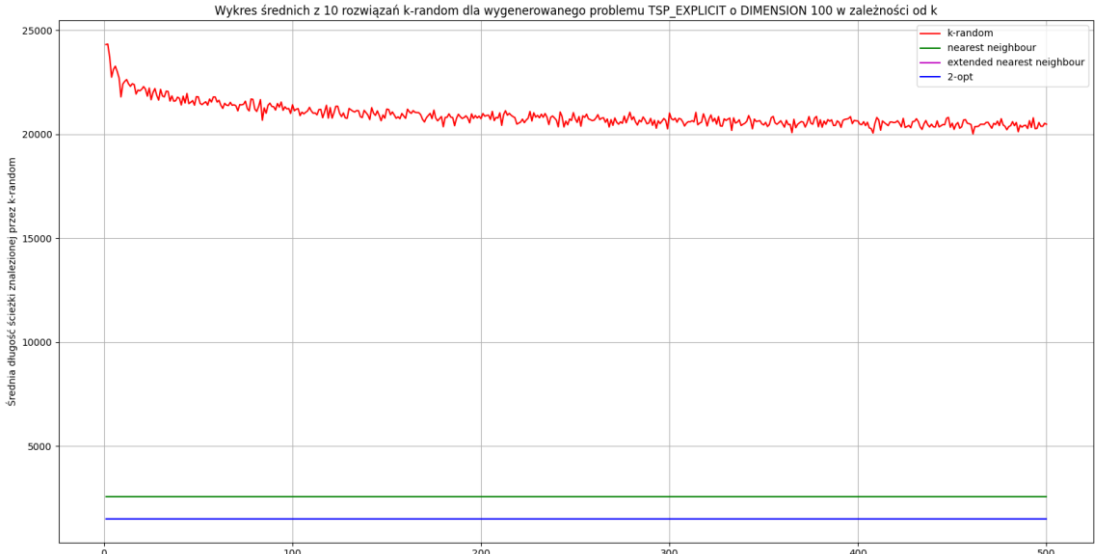


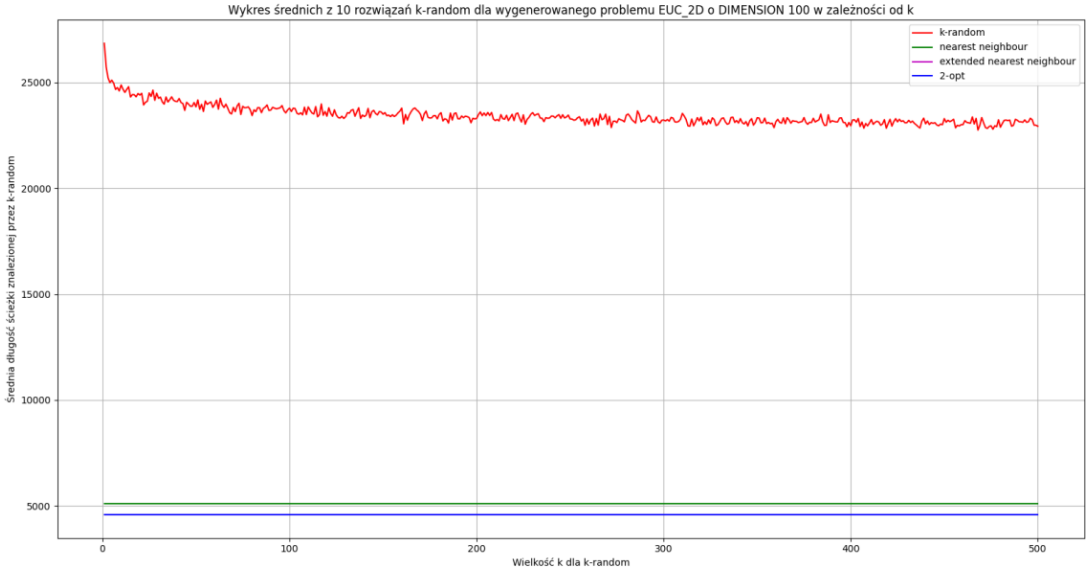




Jak widać dla małego rozmiaru problemu 10, wraz z wzrostem k algorytm k-random zbliża się średnią jakością rozwiązań do pozostałych algorytmów. Dla jednej z wartości k udaje mu się nawet znaleźć rozwiązanie lepsze od pozostałych algorytmów. Dzieje się tak, gdyż liczba możliwych permutacji dla rozmiaru problemu 10 jest względnie mała. Sprawdźmy co się dzieje kiedy zwiększymy rozmiar:







Jak widać dla rozmiaru problemu = 100, wyniki k-random są bardzo oddalone od pozostałych i nie zbliżają się do nich podobnie szybko jak na poprzednich wykresach. Prawdopodobieństwo znalezienia „dobrej” permutacji jest dużo mniejsze.