

Algorymy metaheurystyczne

Mateusz Chęciński, Mateusz Tofil

18 maja 2022

1 Wprowadzenie

Celem tej listy było zapoznanie się z algorytmem przeszukiwania z zabronieniami (z ang. Tabu search).

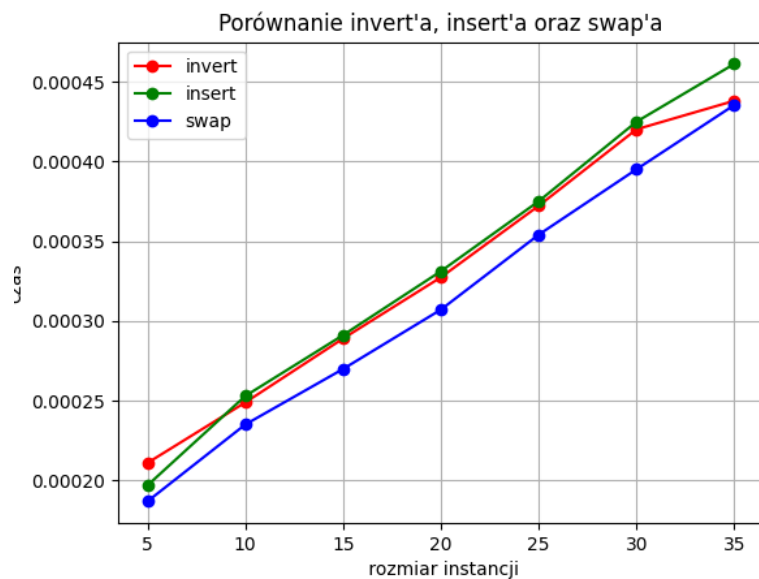
2 Opis algortmu

Zaimplementowany algorytm jest mocno generyczny, mamy wiele opcji do wyboru. Na początku decydujemy, czy podajemy rozwiązanie początkowe (lub np. funkcję heurystyczną, które takie wygeneruje), czy jest ono losowe. Są także 3 możliwe warunki zakończenia: liczba iteracji, liczba iteracji bez zmiany, czas. Algorytm w pętli generuje sąsiedztwo aktualnego rozwiązania (także są 3: swap, insert, invert). Następnie usuwa te rozwiązania, które są na liście tabu (jej długość także jest parametrem) i wybiera najlepsze z pozostałych. Oczywiście może zdarzyć się sytuacja, w której wszystkie rozwiązania są zabronione. Wtedy mamy 5 opcji: 1) zakończenie algorytmu z aktualnym rozwiązaniem 2) usuwanie rozwiązań z listy tabu dopóki jakieś będzie dozwolone 3) powtórzenie algorytmu z innym rozwiązaniem początkowym 4) skorzystanie z kryterium aspiracji (pamięci długoterminowej), tzn. z zapamiętanego wcześniej rozwiązania, które było najlepsze, ale było zabronione, algorytm wykonuje nawrót 5) generowanie nowych sąsiadów dopóki któryś będzie dozwolony

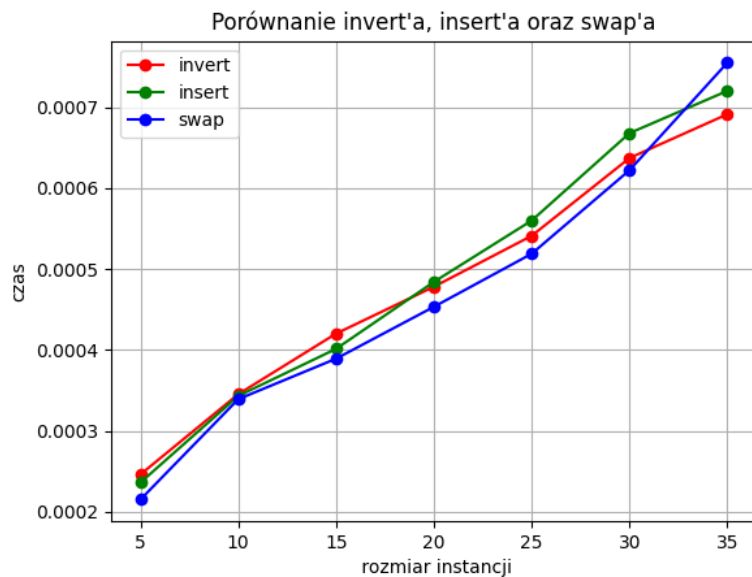
Język programowania: Python 3.10

3 Porównanie otoczeń: insert, invert, swap

W celu zbadania jakie otoczenie jest najlepsze z powyższych 3, przeprowadziliśmy eksperymenty wywołując metode ze zmienionym parametrem początkowym. Wszystkie eksperymenty były przeprowadzone dla tej samej instancji wraz z tym samym rozwiązaniem początkowym. Wykres poniżej został wykonany dla intacji o macierzy pełnej.



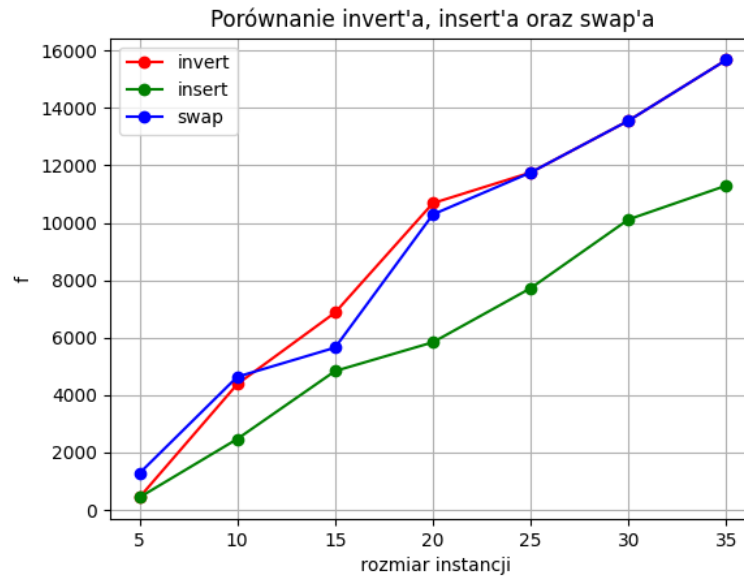
Następnie przeprowadziliśmy ten sam eksperyment, z tą różnicą że teraz dla przestrzeni euklidesowej.



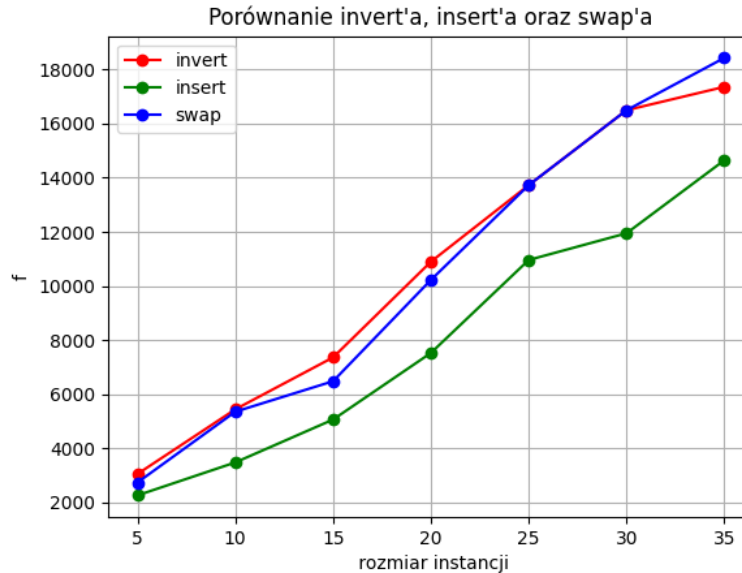
Jak możemy zauważyć, dla wszystkich przeprowadzonych przez nas instancji, otoczenie *swap* okazało się być najelipszym. Pozostałe otoczenia, też nie są złe. Wszystkie otoczenia działają w czasie $\mathcal{O}(n)$ i różnią się tylko stałą, najmniejszą stałą posiada *swap*

Poprzednio porównaliśmy jak czas działa dla poszczególnych otoczeń. Teraz porównamy jak wybór otoczenia wpływa na koszt cyklu dla tych samych instancji. Ponownie wykresy, przygotowaliśmy dla instancji pełnej ograniczeniami dla przestrzeni euklidesowej. Wykresy w odpowiedniej kolejności zostały wstawione poniżej.

Dla instancji o macierzy pełnej, wygenerowanej losowo.



Dla instancji z macierzą, wypełnioną odległościami z przestrzeni euklidesowej.

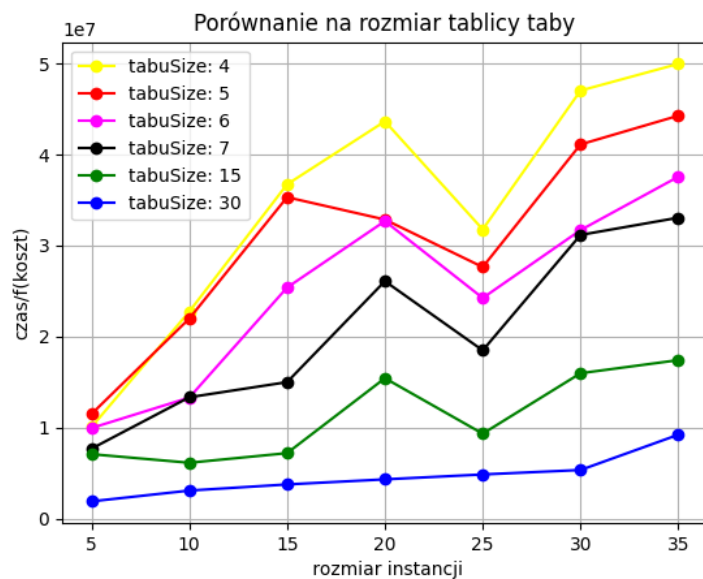


Tym razem *swap* nie był najlepszym wyborem z możliwych przez nas zaimplementowanych otoczeń. Teraz najlepszy okazał się *swap*. Wybierając właśnie to otoczenie dla przebadanych instancji, mogliśmy się spodziewać funkcji kosztu o najniższej wartości.

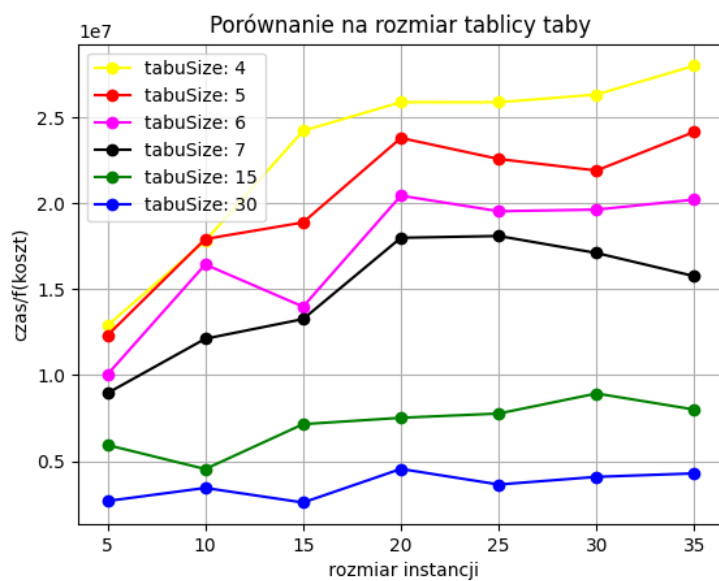
4 Czy długość listy Tabu ma znaczenie?

Chcielśmy upewnić się, czy zwiększając długość listy Tab'u otrzymamy lepszy wynik, tj cykl o najmniejszym koszcie w chwili kończenia algortmy. Doskonale wiemy, że jakbyśmy zwiększyli sam rozmiar listy tabu to algorytm wykonywał by się znacznie dłużej. Dlatego zdecydowaliśmy się, że podzielimy wartość (koszt) cyklu na zakończeniu algorytmu przez czas w jakim otrzymaliśmy wynik.

Wykres dla instancji z macierzą wygenerowaną losowa, pełną.



Wykres dla przestrzeni euklidesowej.



Wyszło, tak jak zakładaliśmy, czyli mimo, że algorytm dłużej pracował, (im większa liczba tabu, tym dłużej działa), daje lepsze wyniki. Zatem warto jest poczekać odpowiednio dłużej aby otrzymać lepszy wynik. Pytanie tylko jak długo opłaca się czekać? Dla listy tabu długości 7, obiecujący wynik dostajemy adekwadnie szybko wraz z "dobrym" rezultatem. Nie musimy czekać dwa razy

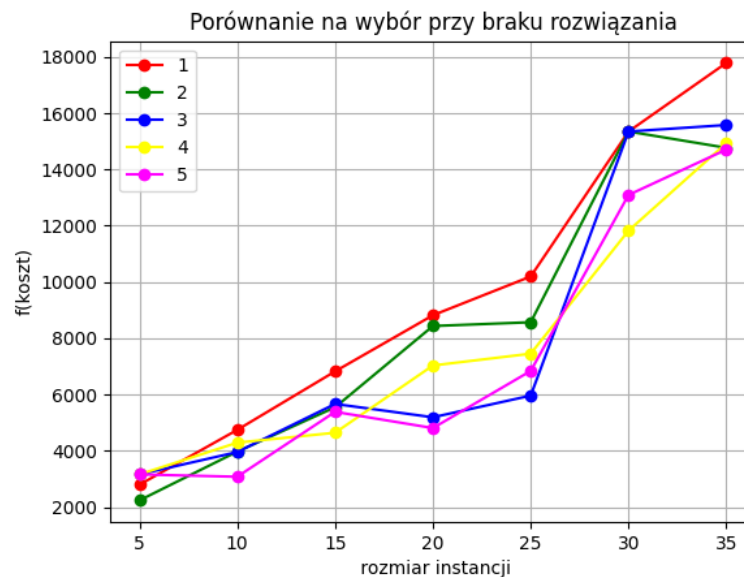
dłużej, niż w przypadku listy np. dł. 30, a wynik nie różni się znacząco między nimi dwoma.

5 Jak decyzja o braku obiecującego rozwiązania z $N(\pi)$ wpływa na koszt?

Zaimplementowaliśmy wszystkie możliwe kroki, które należy podjąć jak wybór obiecującego rozwiązania w otoczeniu są zabronione. Podobnie jak na liście zadań wprowadziliśmy następujące oznaczenia. Przez:

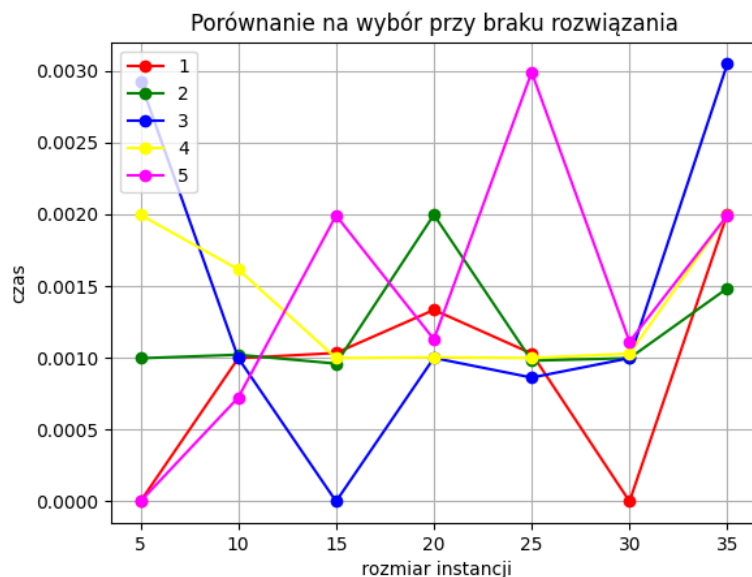
1. -> Zakończenie działania algorytmu
2. -> Stopniowe odrzucanie najstarszych reprezentacji z listy tabu tak długo aż co najmniej jedno rozwiązanie nie jest zabronione
3. -> Dokonanie restartu TS z innego rozwiązania początkowego
4. -> Przejsie do rozwiązań z listy nawrotów, czyli listy obiecujących rozwiązań tworzonych w trakcie działania algorytmu.
5. -> Tymczasowe skorzystanie z innego, szerszego otoczenia.

Z zaimplementowanych przypadków, które należy podjąć przy braku obiecującego rozwiązania z $N(\pi)$, porównaliśmy jak wybór wpływa na funkcję kosztu.



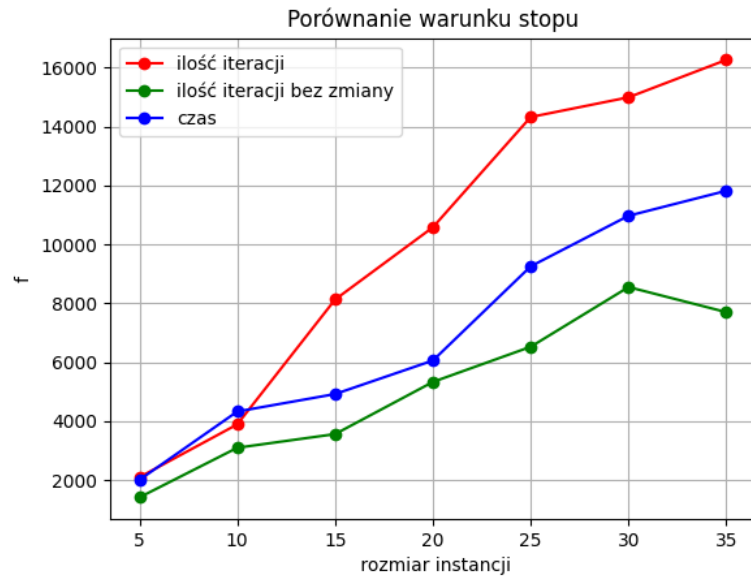
Jak można się było spodziewać, gdy brakuje obiecującego rozwiązania i kończymy z aktualnym najlepiej odkrytym rozwiązaniem, funkcja kosztu od niego nie będzie jakaś bardzo obiecująca i praktycznie zawsze będzie dawała gorsze

rezulaty niż np. użycie listy nawrotów czy odrzucanie starszych reprezentacji z listy tabu. Dokonanie restatru algorytmu jest oparte losowością, ciężko jest stwierdzić czy przynosi zawsze lepszy wynik. Możemy cały czas trawiać "dobrze" rozwiązania początkowe, które będą prowadziły jednoznacznie do optymalnego kosztu, a może się zdarzyć że będziemy wybierać przypadki skrajne i wyniki będą dosyć oddalone od optymalnego rozwiązania.



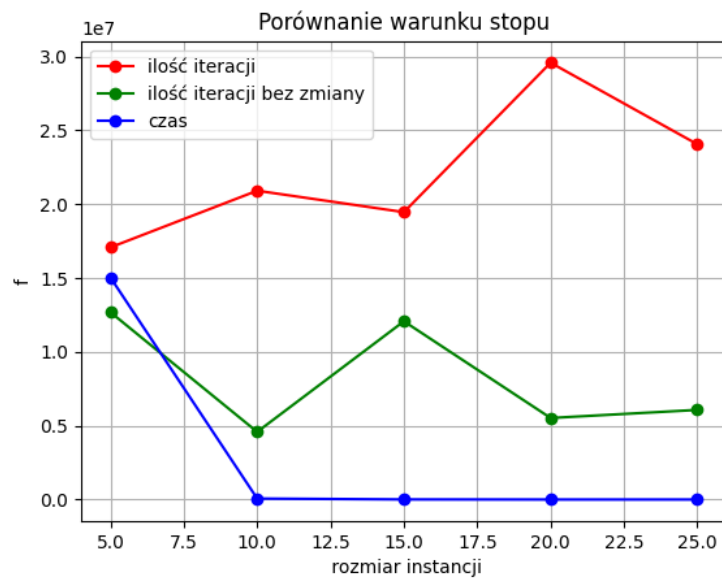
6 Warunek stopu

Zaimplementowaliśmy trzy różne warianty zatrzymania się algorytmu. Pierwszy z nich zaznaczono na czerwono na poniższym wykresie jest ilość ogólnej iteracji algorytmu. Drugi z kolei, na zielono, ilość iteracji algorytmu, gdzie nie znaleźliśmy lepszego rozwiązania, przez k-iteracji. Ostatnim z kolei jest, czas, zaznaczony kolorem niebieskim.



Funckja kosztu dla ograniczenia samymi iteracjami jest znacząco większa niż liczenie iteracji bez zmian. Jest to naturalne zachowanie, wynikające z faktu, że prawie zawsze

$$I + K = \text{iteracje}(\text{iloscBezZmian}) > \text{iteracje}(\text{zwykle}) = I$$



Jak można było się spodziewać, ilość iteracji bez zmiany zdecydowanie korzystniej wpływa na funkcję kosztu niż sama ilość iteracji. Teraz porównamy, jak warunek stopu wpływa na efektywność optymalnego rozwiązania. Jeżeli wybierzemy odpowiednio duży czas, to algorytm będzie tak długo pracował, aż znajdzie optymalne rozwiązanie. Inaczej jest trochę z ograniczeniami na ilość iteracji. Można było się spodziewać, że w wariancie, gdzie liczby iteracji bez zmian, współczynniki efektywności będzie znacznie lepszy od samego liczenia iteracji. Jest to oczywiste, bo iteracje wykonujemy za każdym razem, a iteracji bez zmian wykonamy dodatkowo więcej, na koniec działania algorytmu, niż samo liczenie iteracji.

7 Złożoność obliczeniowa Tab'u Search

Złożoność obliczeniowa zależy od wielu czynników takich jak: rozmiar sąsiedztwa, wybór otoczenia, długość listy taby.

Eksperymentalnie wyznaczyliśmy, że algorytm jest $\mathcal{O}(n^2)$

