Raport z projektu
Optymalizacja dyskretna – problem marszrutyzacji

Dorota Rzewnicka, Oliwia Trzcińska, Jakub Niemyjski, Jan Kwiecień

Maj 2025

1 Cel

Praca ta ma na celu rozwiązanie problemu marszrutyzacji (ang. vehicle routing problem). Jest to jednak zagadnienie zaliczane do NP-trudnych, więc przydatne będą tu heurystyki. W tej pracy przedstawione zostaną rozwiązania przy pomocy algorytmu symulowanego wyżarzania oraz algorytmu genetycznego. W efekcie zaprezentowane będą przykłady działania takich algorytmów dla zadanych argumentów wejściowych. Dodatkowo przeprowadzone zostaną testy, w jaki sposób w danych sytuacjach schładzanie, a więc obniżanie temperatury jest najkorzystniejsze pod względem szybkości zbieżności algorytmu do minimum. Porównamy także ze sobą oba implementowane algorytmy.

2 Opis problemu

Jesteśmy firmą logistyczną posiadającą flotę samochodów dostawczych i jedną bazę, z której rozwozimy przesyłki o różnej wielkości do naszych klientów. Każdego dnia stajemy przed problemem, ile wysłać pojazdów z naszej floty oraz jakie im wyznaczyć trasy, tak, aby funkcja kosztu dla danego rozwiązania była najmniejsza.
Obowiązują jednak ograniczenia. Każdy pojazd z naszej floty ma ograniczoną pakowność Q (każdy taką samą) oraz ograniczony czas pracy D (każdy taki sam). Ponadto klient musi być obsłużony wyłącznie przez jeden pojazd z floty.
Jako dane wejściowe dostajemy tablicę współrzędnych bazy i punktów dostawy oraz wektor d wymiaru $1 \times m$, gdzie m to ilość klientów do obsłużenia, w którym d_i reprezentuje wielkość dostawy do i-tego klienta.

3 Algorytm symulowanego wyżarzania

3.1 Model i zarys rozwiązania

Jako łańcuch Markowa w algorytmie symulowanego wyżarzania proponujemy łańcuch, którego przestrzeń stanów stanowią możliwe realizacje dostarczenia klientom ich pakunków. Mianowicie liczy się nie tylko informacja, do którego klienta dojeżdża który pojazd, ale również kolejność dostarczania przesyłek przez pojazdy. Proponowanymi stanami, do których łańcuch może przejść w jednym kroku są takie, które:

- różnią się albo zamianą kolejności obsługi wybranej w sposób losowy (jednostajnie) pary na trasie pewnego, dokładnie jednego pojazdu z floty,
- powstały z obecnego w ten sposób, że zamieniono dostawcę dla pewnego, losowo wybranego klienta (wybór każdego klienta jest jednakowo prawdopodobny) i tegoż klienta wstawiono jako ostatnią przesyłkę nowego przewoźnika jego przesyłki.

W problemie VRP istnieje kilka możliwości wyboru funkcji kary do optymalizacji. W tym projekcie zdecydowaliśmy się na

$$f(x, y) = \alpha x + \beta y,$$

gdzie α, β to współczynniki, stałe proporcjonalności, które mogą być wybrane wedle uznania, zależnie od potrzeb, x oznacza liczbę pojazdów finalnie wysłanych w drogę, a y sumę odległości pokonanych przez wszystkie pojazdy. Zauważmy, że są to realne założenia kosztów jakie musimy ponieść, ponieważ zmienna x oznaczałaby konieczność opłacenia tylu właśnie kierowców, a zmienna y opłatę za paliwo. Dodatkowo, zakładając w przybliżeniu, że kierowcy jeżdżą wszędzie ruchem jednostajnym i każdy kierowca jeździ taką samą prędkością, to ze wzoru $v = \frac{s}{t}$ widać, że droga przebyta jest wprost proporcjonalna do czasu trwania ruchu, a więc można również interpretować y jako czas jazdy danego kierowcy.

Skrypt został utworzony z wykorzystaniem języka Python. Funkcjonalności zostały skomponowane w funkcje, których opis zostanie poniżej przybliżony. Pseudokod docelowego algorytmu rozwiązującego postawiony problem zamieszczono poniżej, Mimo tego, że nie jest to precyzyjny opis, to właściwie obrazuje koncepcje i intuicje za nim stojące.

Algorithm 1 Algorytm symulowanego wyżarzania

Require: temperatura początkowa $T > 0$, współrzędne bazy i klientów w $\mathbb{R}^2$, `liczba_iteracji`$\in \mathbb{N}_+$, $d \in \mathbb{R}^n_+$, $Q \geqslant 0$, $D \geqslant 0$, $\alpha \geqslant 0$, $\beta \geqslant 0$

1: **for** $i \leftarrow 1, \ldots,$ liczba klientów **do**
2: generuj losowo możliwe przyporządkowanie pojazdów do klientów oraz ich trasy spełniające ograniczenia o ładowności i czasie pracy dysponując co najwyżej i pojazdami
3: **for** $i \leftarrow 1, \ldots,$`liczba_iteracji` **do**
4: zaproponuj nowe przyporządkowanie pojazdów
5: **if** nowe przyporządkowanie spełnia warunki zadania **then**
6: **if** wartość funkcji kosztu dla nowego przyporządkowania jest niższa niż obecna **then**
7: zmień aktualny stan na nowy, proponowany
8: **else**
9: **if** wartość funkcji kosztu dla nowego przyporządkowania jest wyższa niż obecna **then**
10: z pewnym prawdopodobieństwem przejdź do stanu proponowanego, o wyższej wartości kosztów
11: **end if**
12: **end if**
13: **end if**
14: zmniejsz temperaturę T
15: **end for**
16: rozwiązanie w tym kroku zapamiętaj w liście optymalnych rozwiązań
17: **end for**
18: z listy optymalnych rozwiązań znajdź takie, które ma najmniejszą wartość kosztu

Jak widać, niemalże na każdym etapie algorytmu jest wiele rzeczy do uściślenia.

3.2 Szczegółowy opis implementacji algorytmu

W linii 2. algorytmu, każdemu klientowi przyporządkowany zostaje jeden pojazd ze zbioru pojazdów oznakowanych numerami $1, 2, \ldots, i$. Jest to realizowane w funkcji `stworz_wektor_po-jazdy`, która jako parametry wejściowe przyjmuje liczbę klientów oraz wspomnianą przed chwilą liczbę i. Zanim jeszcze ustalimy kolejność obsługi klientów przez każdego przewoźnika, należy sprawdzić, czy w zupełnie losowy sposób wygenerowane przyporządkowanie spełnia warunki zadania dotyczącego pakowności każdego z pojazdów. Zostało to zaimplementowane w funkcji `sprawdz_ladownosc`. Najpierw wprowadza się kontrolę pakowności, ponieważ dla dowolnej kombinacji tras takiego przyporządkowania pojazdów do klientów,

warunek o pakowności zostaje niezmienniczy. Gdy warunek okazuje się nie być spełniony, generujemy jeszcze raz w sposób zupełnie niezależny od poprzedniego kolejne przyporządkowanie. W przypadku, gdy odpowiednio dużą liczbę razy warunek pakowności nie zostanie spełniony, algorytm dochodzi do wniosku, że przy takich warunkach, znalezienie rozwiązania jest dla niego niemożliwe. Gdy warunek pakowności jest jednak spełniony, na tej samej zasadzie, przy już ustalonym przyporządkowaniu klientów do pojazdów, generujemy losową kolejność obsługi klientów w obrębie pojazdów. Do tego celu służy funkcja `potasuj`. Aby skontrolować odległości pokonywane przez pojazdy wprowadzono funkcję `euclidean_dist` mierzącą odległości euklidesowe między punktami w $\mathbb{R}^2$ oraz funkcję `odleglosci`, która przyjmując współrzędne położenia bazy i klientów zwraca macierz odległości między nimi. Powyżej skonstruowane funkcje pozwoliły na skontrolowanie poprawności danych tras w nowo utworzonych funkcjach `liczenie_odleglosci` i `sprawdzanie_odleglosci`.

W linii 4. algorytmu należy dokonać przejścia między stanami. Implementacją pierwszej wymienionej metody jest funkcja `reorder`, a implementacją drugiej `swap` wraz ze swoją metodą pomocniczą `losuj_liczbe_bez_srodka`, która decyduje, do której ciężarówki zostanie przyporządkowany klient, który zmienia pojazd. Wybór między metodami jest w pełni losowy, standardowo jednakowo prawdopodobne jest wywołanie funkcji `reorder` jak i `swap`. Jeżeli niepokojąco dużo razy funkcja nie będzie znajdować możliwego proponowanego stanu do przejścia z uwagi na ograniczenia, wewnętrzna pętla skończy się przedwcześnie.

Enigmatyczne sformułowanie w linii 10. brzmiące „z pewnym prawdopodobieństwem" oznacza prawdopodobieństwo określone w algorytmie symulowanego wyżarzania, tzn.

$$\exp\left(-\frac{1}{T}\left(f(l)-f(k)\right)\right),$$

przy czym k jest aktualnym stanem, a l jest stanem proponowanym zgodnym z warunkami zadania.

Cała objaśniona powyżej procedura dzieje się w zewnętrznej pętli, która rozpoczynana jest generacją innego rozwiązania początkowego i wzięcie najmniej kosztownego rozwiązania ze wszystkich uzyskanych w tejże pętli. Wykonano tutaj więcej niż jeden raz klasyczny algorytm symulowanego wyżarzania z uwagi na to, że istnieje szansa na utkwienie algorytmu w lokalnym minimum, a przy wielu wyborach stanów początkowych, które to najprawdopodobniej różnią się między sobą chociażby ilością pojazdów, szansa na znalezienie globalnego minimum rośnie.

3.3 Szukanie optymalnych parametrów

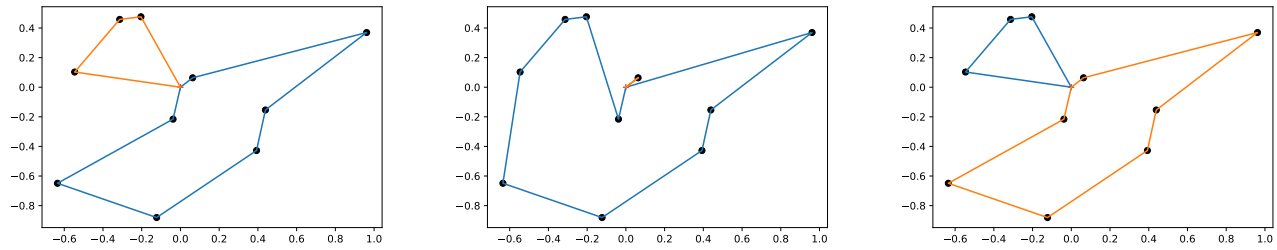
Kolejnym etapem naszej pracy będzie badanie jakości uzyskanych wyników w zależności od tempa zmniejszania temperatury. Będzie nam to potrzebne do wyboru optymalnego rozwiązania, które następnie skonfrontujemy z optymalnym rozwiązaniem algorytmu genetycznego. Rozważymy cztery typy schładzania:

- **geometryczne** - w każdej iteracji temperatura jest przemnażana przez przyjętą wcześniej stałą $\gamma \in (0,1)$. W tym przypadku im mniejsza jest temperatura, tym schładzanie zachodzi wolniej;
- **logarytmiczne** - w k - tej iteracji temperatura jest mnożona zgodnie ze wzorem: $\frac{\ln(1+k)}{\ln(2+k)}$;
- **liniowe** - temperatura zmniejszana jest liniowo, tj. zawsze o tę samą stałą wartość T_x ($T_{k+1} = T_k - T_x$);
- **harmoniczne** - w każdej iteracji mnożymy temperaturę przez $\frac{1}{k}$, gdzie k - numer iteracji;

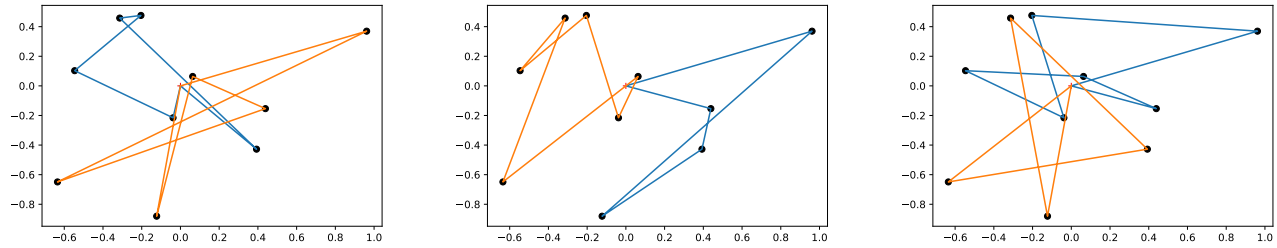
Za każdym razem będziemy analizować ten sam przypadek, tj. dla ustalonych współrzędnych bazy, klientów, pojemności pojazdów i czasu pracy. Niezmienione pozostaną również współczynniki α oraz β a także wektor d_m reprezentujący wielkość zamówienia m - tego klienta. W schładzaniu geometrycznym ustalimy współczynnik γ na 0,95 (zalecana wartość w literaturze), a w schładzaniu liniowym T_x wyniesie wartość temperatury początkowej podzielonej przez liczbę iteracji, tak aby w ostatniej iteracji temperatura wyniosła 0. Ograniczymy się jedynie do dwóch poruszających się ciężarówek (zadbamy o to, aby były

w stanie rozwieźć wszystkie przesyłki). Liczba iteracji wyniesie 10000, a temperaturę zbadamy w trzech zakresach: 1000, 10000, 100000 (być może któreś schładzanie zadziała lepiej dla innej wartości temperatury początkowej).

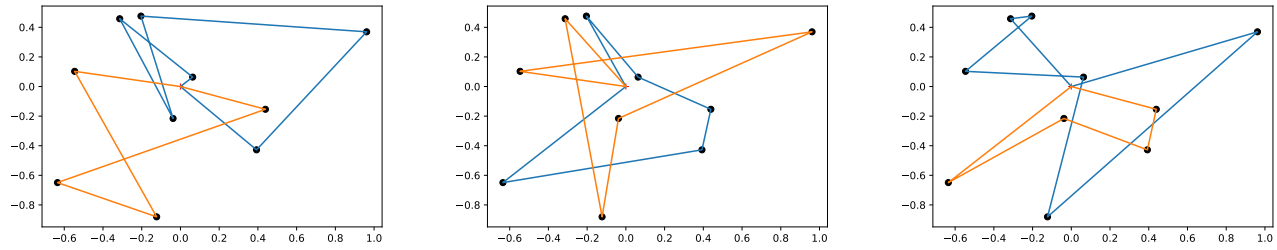
Uzyskane wyniki będziemy porównywać poprzez funkcję kosztu. Wyznaczone trasy prezentują się następująco:



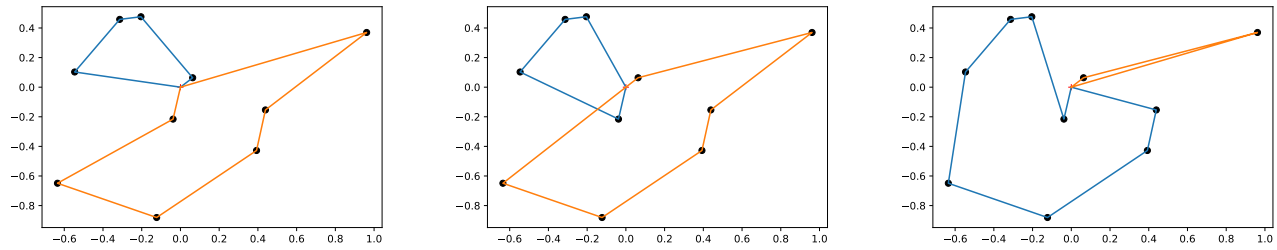
Rysunek 1: Wyniki dla schładzania geometrycznego i temperatur odpowiednio 1000, 10000 oraz 100000.



Rysunek 2: Wyniki dla schładzania logarytmicznego i temperatur odpowiednio 1000, 10000 oraz 100000.



Rysunek 3: Wyniki dla schładzania liniowego i temperatur odpowiednio 1000, 10000 oraz 100000.



Rysunek 4: Wyniki dla schładzania harmonicznego i temperatur odpowiednio 1000, 10000 oraz 100000.

W przypadku schładzania geometrycznego funkcja kosztów wynosiła nie więcej niż 20. Podobnie sprawa miała się dla harmonicznej wersji obniżania temperatury, gdzie funkcja kosztów nie przekroczyła wartości 22. Dużo gorzej poradziły sobie środkowe warianty. Tutaj wyniki oscylowały w okolicach 27 i 30. W przypadku liniowego ostudzania, zachodzi ono zbyt wolno, przez co algorytm nie ma szansy się ustabilizować. W dodatku podobny wynik dla pierwszej i czwartej metody może być spowodowany podobnym tempem spadku, tzn. szybko na początku i wolno w ostatnich etapach algorytmu. Niemniej możemy zauważyć, że dobór metody schładzania może być kluczowy w jakości działania algorytmu symulowanego wyżarzania.

Przedstawimy teraz powyższe zależności na dwóch wykresach. Pierwszy z nich będzie obowiązywał dla współrzędnych klientów i bazy rozważanych w poprzednim przykładzie. Drugi natomiast, uwzględni współrzędne miast Polski przeskalowanych do kwadratu $[-1, 1] \times [-1, 1]$. Będą to: Płock, Sochaczew, Kurzętnik, Niemyje-Ząbki, Paryż, Wielka Lipa, Wenecja, Bagdad, Rzym i Swornegacie. Współrzędne bazy to współrzędne miejscowości Piątek. Do odpowiedniego przeskalowania posłuży nam funkcja *scale_points*. Wartości funkcji kary w zależności od temperatury początkowej i tempa schładzania dla zboru danych pierwszego wyglądają następująco:

Rysunek 5: Wartości funkcji kary dla zbioru odpowiednio pierwszego i drugiego w zależności od temperatury początkowej i rodzaju schładzania.

Widzimy, że najlepiej poradziło sobie schładzanie geometryczne, a tuż za nim harmoniczne. Widzimy, że w pierwszym przypadku schładzanie liniowe poprawiało się, wraz ze wzrostem temperatury. Mimo tego schładzanie liniowe i logarytmiczne okazały się dość niestabilne. Z powyższych przyczyn przy porównywaniu algorytmów posłużymy się schładzaniem geometrycznym.

4 Algorytm genetyczny

4.1 Model i zarys rozwiązania

W tej części przedstawimy drugi sposób rozwiązania problemu marszrutyzacji. Algorytm genetyczny, inspirowany procesami ewolucji biologicznej, operuje na populacji możliwych rozwiązań, które ewoluują w kolejnych pokoleniach dzięki zastosowaniu mechanizmów selekcji, krzyżowania i mutacji.
Podstawą działania algorytmu genetycznego jest operowanie na populacji rozwiązań, które z pokolenia na pokolenie ewoluują w kierunku coraz lepszych rozwiązań problemu marszrutyzacji. Każde rozwiązanie (tzw. chromosom) jest reprezentowane jako permutacja genów, gdzie genem może być klient (oznaczony parą (i, d_i), gdzie i to numer klienta, a d_i – wielkość dostawy) lub symboliczny znacznik `'truck'`, wyznaczający początek trasy nowego pojazdu.
Zbiór genów zawiera wszystkich klientów oraz $n - 1$ znaczników `'truck'` (gdzie n to liczba dostępnych pojazdów). Jeden chromosom to więc permutacja długości $m+n-1$, gdzie m to liczba klientów. Dekodowanie chromosomu oznacza podział na trasy, z których każda rozpoczyna się i kończy w bazie.
Aby umożliwić ocenę rozwiązań, zaimplementowano funkcje sprawdzające poprawność chromosomów

względem ograniczeń ładowności (`is_capacity_allowed`) oraz maksymalnego dystansu (`is_distance_allowed`). W obliczeniach odległości wykorzystano metrykę euklidesową, a odległości między punktami są przechowywane w postaci macierzy.
Funkcja kary, tak samo jak w poprzednim algorytmie, ocenia chromosom według wzoru:

$$f(x, y) = \alpha x + \beta y,$$

gdzie x to liczba użytych pojazdów, a y to łączny dystans przejechany przez wszystkie pojazdy, liczony na podstawie odległości pomiędzy punktami trasy każdego pojazdu.
Przedstawimy teraz pseudokod zaimplementowanego przez nas algorytmu.

Algorithm 2 Algorytm genetyczny

Require: współrzędne bazy i klientów w $\mathbb{R}^2$, liczba iteracji `n_gen` $\in \mathbb{N}^+$, liczba osobników w populacji `size` $\in \mathbb{N}^+$, liczba pojazdów $n \in \mathbb{N}^+$, pojemność pojazdu $Q > 0$, maksymalny dystans trasy pojazdu $D > 0$, współczynniki $\alpha, \beta \geqslant 0$, prawdopodobieństwo mutacji `prob_mutate` $\in (0, 1)$, proporcja krzyżowań `ratio_cross` $\in (0, 1)$, rozmiar turnieju selekcyjnego $k \in \mathbb{N}^+$

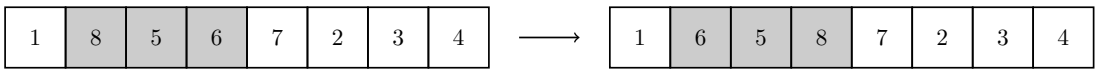
1: generuj populację początkową losowych chromosomów o rozmiarze `size`
2: wyznacz liczności elity oraz grupy osobników do krzyżowania
3: **for** $i \leftarrow 1, \ldots,$ `n_gen` **do**
4: uzupełnij grupy (elitę i osobników do krzyżowania), każdorazowo wybierając najlepszy chromosom spośród k losowych z populacji
5: utwórz nowe osobniki poprzez krzyżowanie par
6: dla każdego nowego dziecka zastosuj mutację z prawdopodobieństwem `prob_mutate`
7: połącz osobników wybranych bez zmian (elitę) z nowo powstałymi, tworząc nową populację
8: **end for**
9: wybierz z populacji najlepszy chromosom spełniający ograniczenia ładowności i długości trasy
10: zwróć zdekodowane trasy pojazdów oraz wartość funkcji kosztu dla najlepszego rozwiązania

Na uwagę zasługuje fakt, że ograniczenia ładowności i dystansu sprawdzane są dopiero na etapie ostatniej iteracji. Jest tak dlatego, że po skrzyżowaniu dwóch chromosomów niespełniających warunków, może powstać chromosom spełniający je.

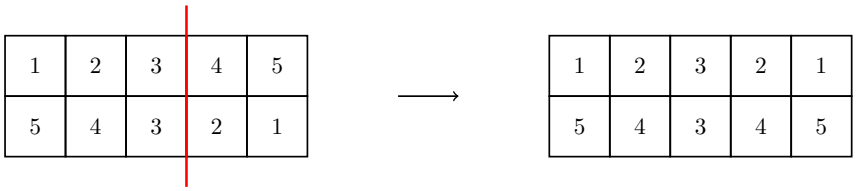
4.2 Opis implementacji

Przedstawimy listę zastosowanych przez nas operatorów wraz z opisem ich działania.

- Operator mutacji:
  Zastosowano klasyczny operator mutacji inwersyjnej: losowo wybierany jest przedział w chromosomie, którego zawartość zostaje odwrócona. Mutacja zachodzi z pewnym zadanym prawdopodobieństwem. Operator ten pozwala na lokalne przeszukiwanie przestrzeni rozwiązań, w szczególności modyfikując kolejność klientów lub pozycję znaczników `'truck'`.



- Operator krzyżowania:
  Krzyżowanie realizowane jest przez prosty *cut-point crossover*: dla dwóch rodziców wybierany jest jeden punkt podziału, po czym wymieniane są fragmenty chromosomów po tym punkcie. Aby uniknąć niepoprawnych rozwiązań (np. zduplikowanych klientów lub zbyt wielu pojazdów), po krzyżowaniu stosowana jest procedura naprawcza. Sprawdza ona powtarzające się geny i dokonuje wymian z drugiego rodzica, dbając o zachowanie unikalności klientów i właściwej liczby pojazdów.

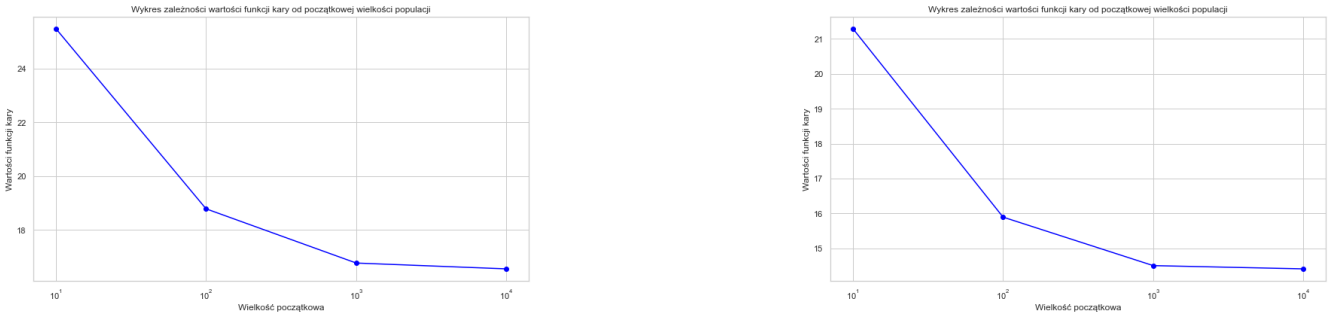


- Selekcja i tworzenie nowej generacji:
  Selekcja rodziców odbywa się metodą turniejową: z populacji losuje się podzbiory rozmiaru k, a do dalszego etapu przechodzą osobniki o najlepszym przystosowaniu w każdej grupie. Nowa populacja tworzona jest z: części osobników (tzw. elita) bez zmian, osobników pochodzących z krzyżowania wybranych rodziców, osobników powstałych po mutacjach dzieci z krzyżowania. Proporcja tych grup sterowana jest parametrem `ratio_cross`, a liczba iteracji (generacji) podana przez `ngen`.

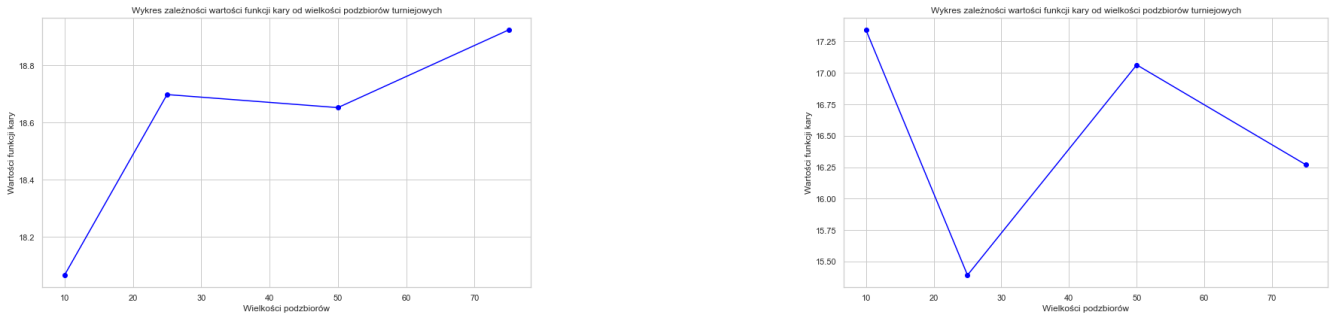
- Główna pętla algorytmu:
  Główna pętla algorytmu wykonuje `ngen` iteracji. W każdej z nich tworzona jest nowa generacja według powyższego schematu. Na końcu wybierany jest najlepszy chromosom spełniający wszystkie warunki (ładowność i długość trasy). Jeśli żaden nie spełnia ograniczeń, algorytm sygnalizuje brak dopuszczalnego rozwiązania.

4.3 Szukanie optymalnych parametrów

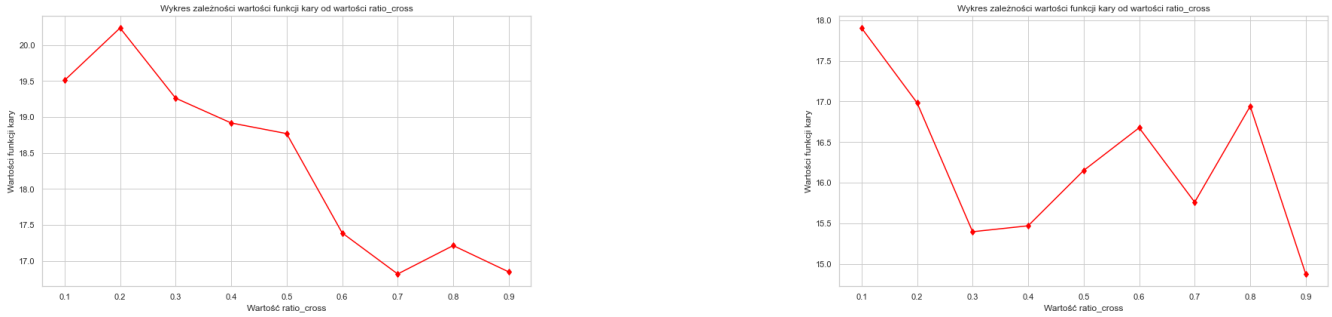
Podobnie jak w symulowanym wyżarzaniu, wybierzemy najbardziej korzystne parametry w celu porównania jakości dwóch algorytmów. Parametr `ngen` będzie stały (równy 10), natomiast przetestujemy `size`, `k`, `ratio_cross` oraz `prob_mutate` odpowiednio dla wartości $[10, 100, 1000,$
$10000]$,$[10, 25, 50, 75]$, $[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$ oraz $[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,$
$0.8, 0.9]$. Do analizy wykorzystamy dwa zbiory użyte poprzednio, tj. losowo wygenerowane punkty i punkty z mapy Polski. W każdym przypadku wynik będzie uśredniony dla 3 wykonań algorytmu. Poniżej przedstawiamy wyniki symulacji.



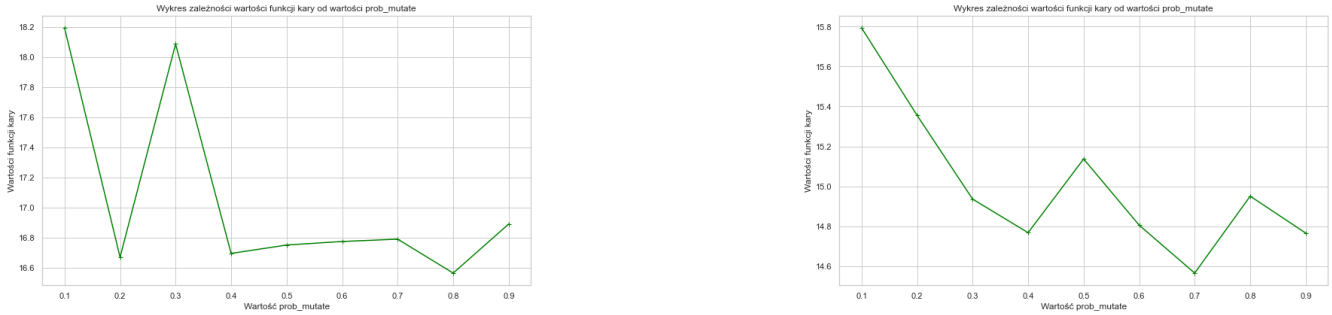
Rysunek 6: Wartości funkcji kary dla zbioru odpowiednio pierwszego i drugiego w zależności od początkowej wielkości populacji.



Rysunek 7: Wartości funkcji kary dla zbioru odpowiednio pierwszego i drugiego w zależności od parametru `k`.



Rysunek 8: Wartości funkcji kary dla zbioru odpowiednio pierwszego i drugiego w zależności od parametru `ratio_cross`.



Rysunek 9: Wartości funkcji kary dla zbioru odpowiednio pierwszego i drugiego w zależności od parametru `prob_mutate`.

Widzimy, że wraz ze wzrostem populacji osiągaliśmy lepsze rozwiązania. Ze względu na zasoby obliczeniowe ograniczymy się do wartości 100. W przypadku parametru `k` jego wartość zależy raczej od zbioru początkowego, zatem wybierzemy wartość 10. Dla `ratio_cross` większe wartości dawały lepsze wyniki, stąd ustalimy jego wartość na 0.9. Podobnie jak dla zmiennej `k`, `prob_mutate` nie wpływają na jakość rozwiązania. W tym przypadku wybierzemy 0.8.

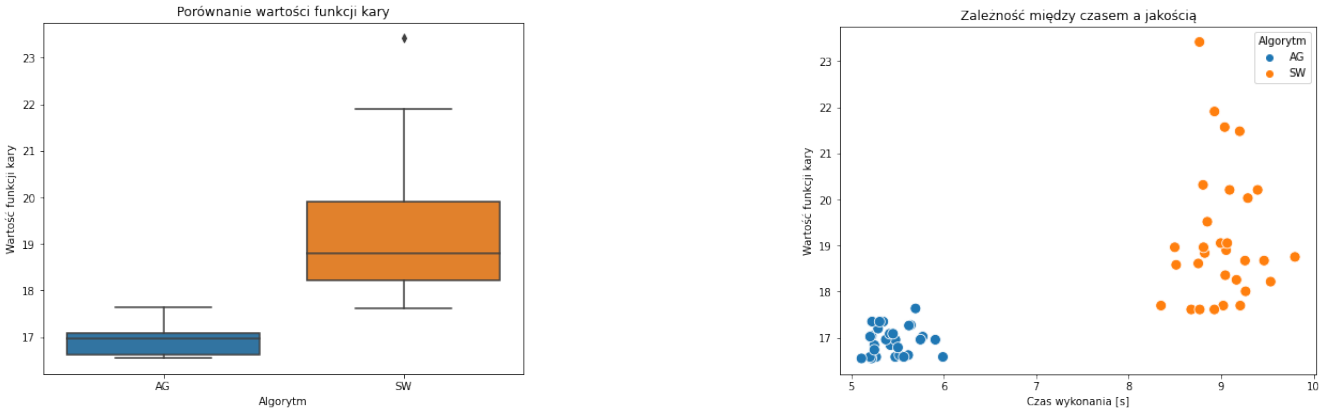
5 Porównanie algorytmów

W tej sekcji porównamy zaimplementowane algorytmy dla ustalonych uprzednio wartości parametrów.

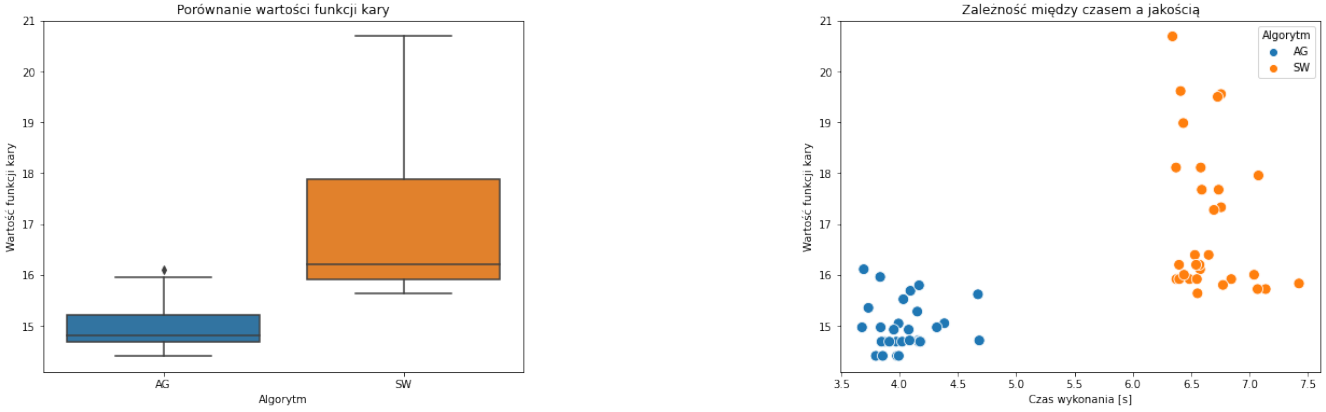
5.1 Wartość funkcji kary oraz czas wykonania

Pierwszym etapem analizy będzie porównanie wartości funkcji kary uzyskiwanych przez badane algorytmy oraz czasu ich wykonania. Przypomnijmy, że rozważane są dwa zestawy danych. Pierwszy z nich zawiera losowo wybrane punkty z kwadratu $[-1, 1] \times [-1, 1]$, reprezentujące współrzędne klientów do odwiedzenia. W tym przypadku baza znajduje się w punkcie $(0, 0)$. Drugi zbiór obejmuje przeskalowane do tego samego kwadratu współrzędne wybranych miejscowości w Polsce, przy czym baza zlokalizowana jest w punkcie odpowiadającym miejscowości Piątek.

Eksperyment polegał na uruchomieniu każdego z analizowanych algorytmów po 30 razy dla obu zbiorów danych. Dla każdego uruchomienia rejestrowano wartość funkcji kary oraz czas wykonania. Na podstawie zebranych danych przygotowano wykresy skrzynkowe ilustrujące rozkład wartości funkcji kary dla obydwu algorytmów. Dodatkowo, utworzono wykresy punktowe przedstawiające zależność pomiędzy wartością funkcji kary a czasem wykonania. Poszczególne algorytmy oznaczono różnymi kolorami.



Rysunek 10: Wartości funkcji kary oraz zależność pomiędzy wartością funkcji kary a czasem wykonania dla obu algorytmów na pierwszym zbiorze.



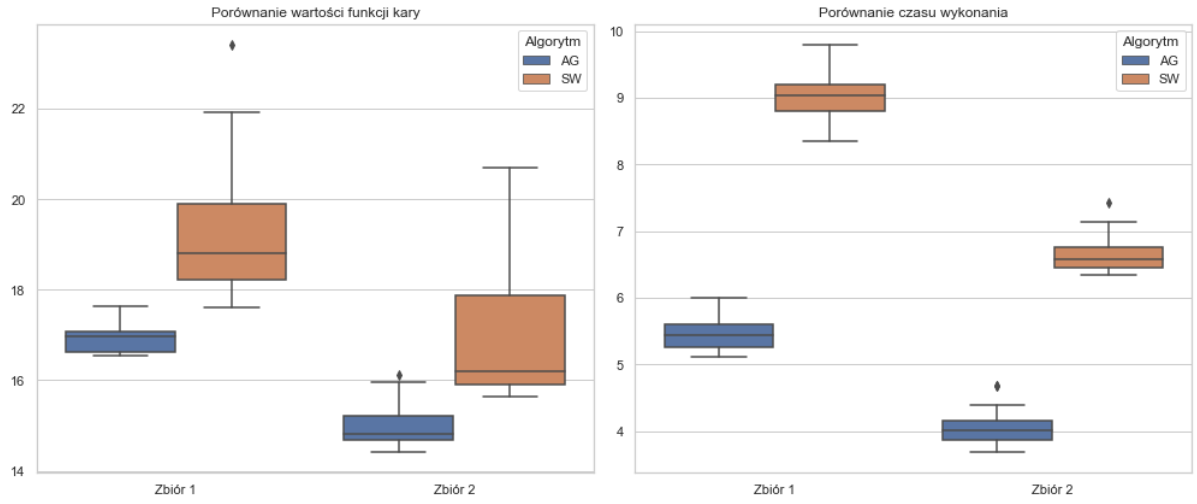
Rysunek 11: Wartości funkcji kary oraz zależność pomiędzy wartością funkcji kary a czasem wykonania dla obu algorytmów na drugim zbiorze.

Widzimy, że na obydwu zbiorach algorytm genetyczny osiągnął lepsze rezultaty, zarówno jeśli chodzi o funkcję kary, jak i czas wykonania.

Kolejne wykresy przedstawiają porównanie tych samych danych jednocześnie dla obydwu zbiorów oraz obu algorytmów. W przypadku zbioru 1 algorytm genetyczny osiąga niższe wartości funkcji kary w porównaniu do symulowanego wyżarzania oraz jego wyniki cechują się mniejszą zmiennością. Dla zbioru 2 różnica ta jest jeszcze bardziej wyraźna.

Wykres po prawej stronie przedstawia porównanie czasu wykonania obu algorytmów. Dla obydwu zbiorów algorytm genetyczny działa szybciej niż symulowane wyżarzanie. W przypadku zbioru 2 czasy wykonania obu algorytmów są nieco krótsze niż dla zbioru 1.

Zarówno pod względem jakości rozwiązania, jak i efektywności obliczeniowej, algorytm genetyczny okazuje się skuteczniejszy od symulowanego wyżarzania na obu analizowanych zbiorach.



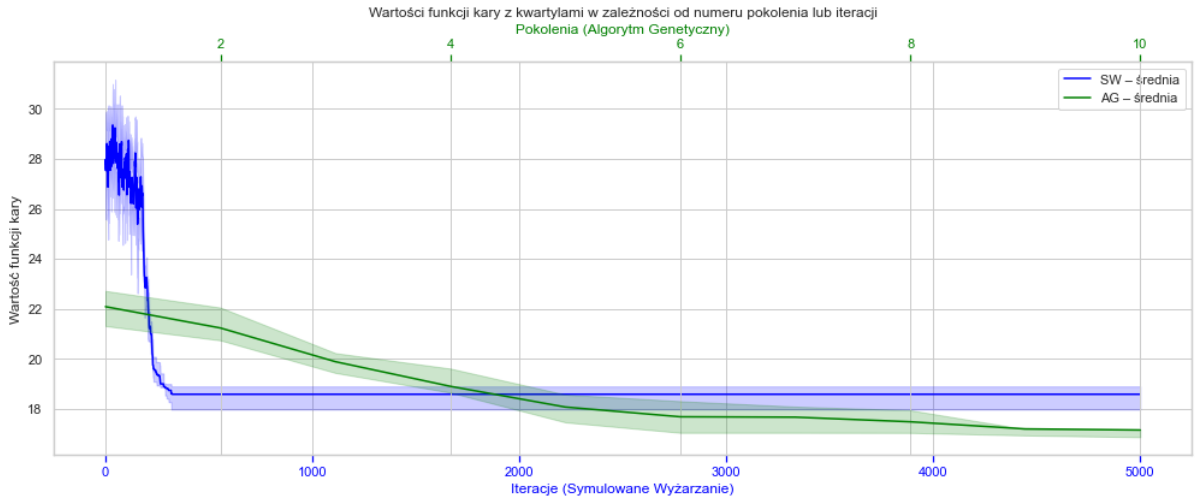
Rysunek 12: Wartości funkcji kary oraz czas wykonania obu algorytmów na obydwu zbiorach danych.

5.2 Szybkość zbieżności

Kolejnym porównywanym elementem będzie szybkość zbieżności funkcji kary dla rozważanych algorytmów.

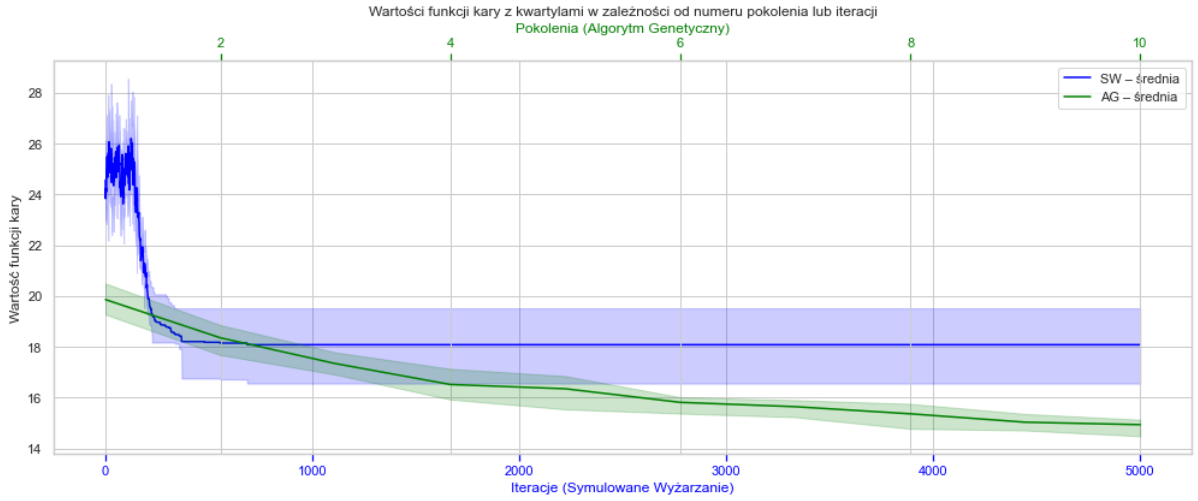
Poniższe wykresy prezentują przebieg zmian wartości funkcji kary w trakcie działania obu metod na danym zbiorze. Ponieważ wyniki poszczególnych uruchomień mogą się istotnie różnić, każdy z algorytmów został uruchomiony 10 razy. Dla każdego uruchomienia rejestrowano wartości funkcji kary w kolejnych iteracjach (dla symulowanego wyżarzania) lub pokoleniach (dla algorytmu genetycznego).

Dla każdej iteracji lub pokolenia obliczono średnią wartość funkcji kary, a także pierwszy i trzeci kwartyl, które posłużyły do wizualizacji rozrzutu wyników. Na tej podstawie powstał wykres z dwiema niezależnymi osiami X: dolna oś przedstawia liczbę iteracji dla symulowanego wyżarzania, natomiast górna – liczbę pokoleń w algorytmie genetycznym. Wartości średnie zostały przedstawione jako linie, a obszar pomiędzy kwartylami został zaznaczony kolorem, ilustrując zmienność wyników pomiędzy uruchomieniami.



Rysunek 13: Zależność osiągniętej wartości funkcji kary w zależności od numeru pokolenia lub iteracji dla pierwszego zbioru.

W przypadku pierwszego zbioru symulowane wyżarzanie w ciągu pierwszych kilkuset iteracji osiąga relatywnie niski poziom funkcji kary, po czym stabilizuje się i utrzymuje ten poziom aż do końca. Dodatkowo, niebieskie pasmo między kwartylami (Q1–Q3) jest bardzo wąskie po fazie początkowej, co świadczy o dużej stabilności i powtarzalności wyników tego algorytmu po osiągnięciu minimum. Dla algorytmu genetycznego widoczny jest stopniowy spadek wartości funkcji kary w kolejnych pokoleniach. Końcowe wartości osiągane przez ten algorytm są lepsze od tych uzyskanych przez symulowane wyżarzanie.



Rysunek 14: Zależność osiągniętej wartości funkcji kary w zależności od numeru pokolenia lub iteracji dla drugiego zbioru.

Wykres dla drugiego zbioru jest podobny. Dodatkowo pokazuje, że symulowane wyżarzanie charakteryzuje się większym rozrzutem wyników między uruchomieniami. Niebieskie pasmo pomiędzy kwartylami pozostaje szerokie praktycznie przez cały czas działania algorytmu. Z kolei algorytm genetyczny wykazuje bardziej stabilne działanie — kwartyle są znacznie węższe, a wartość funkcji kary systematycznie spada.

6 Podsumowanie

W ramach projektu zaimplementowano dwa algorytmy optymalizacyjne: algorytm genetyczny oraz algorytm symulowanego wyżarzania. Wyniki eksperymentów wskazują na przewagę algorytmu genetycznego. Uzyskuje on niższe wartości funkcji kary przy krótszym czasie wykonywania oraz charakteryzuje się większą stabilnością wyników. Ich rozrzut jest zauważalnie mniejszy niż w przypadku symulowanego wyżarzania.

7 Bibliografia

- Yaghout Nouraniy and Bjarne Andresenz, *A comparison of simulated annealing cooling strategies*, J. Phys. A: Math. Gen. 31 (1998) 8373–8385
- https://github.com/fermenreq/TSP-VRP-GENETICS-ALGORITHM/tree/master