ZASTOSOWANIE ALGORYTMÓW EWOLUCYJNYCH

W PLANOWANIU TRAS POJAZDÓW KOMUNALNYCH

SZYMON KUCZATY

MICHAŁ KRZYSZCZUK

Spis treści

Wprowadzenie do problemu....3
Opis analizowanego zadania....3
Przyjęte uproszczenia....3
Model matematyczny....4
Algorytm rozwiązujący problem....5
Opis teoretyczny....5
Implementowana wersja algorytmu....5
Metoda selekcji....5
Operatory krzyżowania....5
Operatory mutacji....6
Uproszczony schemat blokowy....7
Wykorzystywane oprogramowanie....7
Python....7
Biblioteki....8
NumPy....8
Matplotlib....8
GeoPy....8
Inne....9
Testy....9
Problem najkrótszej ścieżki bez załadunku....9
Manualne testy....9
TDD....9
Przybliżona złożoność algorytmu....10
Złożność pamięciowa populacji....10
Złożoność czasowa operatorów....11
Otrzymane wyniki....12
Problem TSP....12
Problem właściwy....13
Instancja I (Gorzyce Small 2)....13
Instancja II (Gorzyce Small)....16
Optymalny dobór parametrów dla instancjii II....16
Instancja III (Gorzyce)....21
Wnioski i ocena algorytmu....21

Wprowadzenie do problemu

Opis analizowanego zadania

Analizowany problem jest fragmentem dyscypliny optymalizacji matematycznej. W wybranym przykładzie przedstawiono zadanie znalezienia w przybliżeniu optymalnego przydziału tras dla pojazdów komunalnych. Mając dany rozmiar zaplecza gospodarczego właściciela (reprezentowany przez flotę samochodów) oraz obligatoryjną listę posesji, w których należy świadczyć usługę odbioru nieczystości założono potrzebę minimalizacji kosztów.

Jednym z pierwszych etpów było określenie jakiego typu jest to zadanie. W związku geometryczną lokalizacją posesjii klijentów pewna część zadania jest związana z poszukiwaniem wektora ścieżek w macierzy odległości, o minimalnej wartości ich sumy totalnej. Fizyczna pojemność skrzyni samochodów oraz wymagane normy obciążenia nakładają na rozwiązania konieczność optymalnego wyboru odwiedzanych punktów przez każdy samochód, aby od momentu startowego do momentu wysypania jego skrzynia zawierała co najwyżej maksymalną dozwoloną ilość ładunku. Kolejną składową problemu jest pytanie, czy wszystkie dostępne samochody powinny być uruchamiane w ramach realizacjii zadania. Pytanie to jest w większości związane z kosztami obsługi pojazdu, kierowcy, pracowników, przeglądów techninych itp.

Przedstawiony problem nie może być aproksymowany żadnym z najczęściej poruszanych problemów optymalizacyjnych (*discrete knapsack problem, travelling salesman problem, allocation problem),* pomimo ich związku z tym zagadnieniem. Postanowiono chwilowo zapomnieć o klasycznych metodach poszukiwania optymalnego rozwiązania czy analitycznych sposobach rozwiązywania zbliżonych zagadnień w celu poznania ciekawych własności i możliwości algorytmów ewolucyjnych.

Przyjęte uproszczenia

- Samochody spalają zawszę taką samoą ilość jednostek pieniężnych/km – niezależnie od obiążenia
- Wszystkie kosze mają równy priorytet, nie istnieje nagroda/kara implikowana kolejnością odbioru koszy
- Odległości między punktami, są mierzone programowo (odległość, między 2 punktami z wykorzystaniem mapy)
- Liczba wyjazdów danego samochodu nie ma górnego ograniczenia, nie przewiduje się jego ewentualnej awarii itp.
- Klijent nie ma do oddania towaru o większym ciężarze niż pojemność jednego samochodu

Model matematyczny

Każdy z osobników populacji reprezentuje pewne dopuszczalne rozwiązanie postawionego problemu oraz posiada zbiór informacji stanowiący **genotyp**, który jest podstawą do utworzenia **fenotypu** – poddawanego ocenie. W naszym modelu oba te pojęcia są równoważne i będą stosowane zamiennie. Genotyp pojedynczego rozwiązania reprezentuje kolejność odwiedzanych węzłów (wierzchołki grafu reprezentują punkty odbioru nieczystości oraz bazy rozładunkowe, krawędzie reprezentują odległości (w jednostkach km)) .

$$X \in R^{(m+n)} = [-1, \ldots, 0], x_i \geq -1 \wedge x_i \in \mathbb{Z}$$

n – liczba klijentów

m – suma zatrudnionych samochodów oraz zjazdów do wysypiska

Liczba powtórzeń elementów dodatnich (reprezentujących klijenta) zawsze wynosi 0, co reprezentuje jednokrotne odwiedzenie każdego klijenta -**zakładamy** , że sytuacja w której klijent ma cieższy towar niż pojemność wozu jest niemożliwa oraz, że każdy klijent musi zostać odwiedzony.
Krytycznym elemntem każdego rozwiązania jest obliczana dla niego wartość **funkcji przystosowania**, która ocenia jak dobre jest zaproponowane rozwiązanie. W klasycznym ujęciu im większa wartość tym rozwiązanie jest bliższe optymalnego. W prezentowanym sposobie wartość funkcji przystosowania jest tożsama z kosztem generowanym przez konieczność odbioru wszystkich nieczystości. Niewielka zmiana projektowa, w postaci dodaniu to funkcji stałej składowej reprezentującej miesięczny zysk z ryczałtu przeprowadza zastosowaną koncepcję w zgodną z klasyczną.

A zbiór wszystkich wariacji o unikalnych wartościan z zakres [1, n]oraz m elementach ,

ze zbioru { -1, 0 }

$f : A \rightarrow \mathbb{R}$, gdzie f jest zdefiniowana wzorem:

$$f(X) = a * \overline{-1} + \sum_{i=1}^{\overline{-1}} \sum_{j=2}^{\text{długość trasy}} M(j-1, j) + \sum_{k=1}^{\overline{-1}} (DL(k) - O(k))^2$$

a-współczynnik kosztu aktywowania wyjazdu samochodu
$\overline{-1}$ - liczba aktywnych samochodów
M(j-1, j)- koszt przejazdu z wierzchołka j-1 do wierzchołka j, wyliczany z macierzy kosztów
DL(k)- dopuszczalna ładowność k-tego samochodu
O(k)-aktualne obciążenie k-tego samochodu

Algorytm rozwiązujący problem

Opis teoretyczny

Algorytm ewolucyjny przeszukuje przestrzeń rozwiązań problemu w celu wyszukania najlepszych. Sposób działania tej rodziny algorytmów przypomina zjawisko ewolucji biologicznej. Operuje na określonej populacji (o pewnej liczności) jednostek oraz numerze oznaczającym numer generacji. W wydaniu klasycznym algorytm nie wykorzystuje wiedzy o rozwiązywanym problemie. Umiejętność znalezienia dobrych rozwiązań wynika z tego,że do kolejnych populacji awansują jedynie osobnicy najlepsi, oraz ci utworzeni na ich podstawie z wykorzystaniem operatorów tworzących potencjalnie lepiej przystosowane jednostki.

Implementowana wersja algorytmu

Metoda selekcji

Zadanie selekcji polega na wyborze osobników, które zostaną wybrane do grupy rozrodczej (operandy krzyżowania) tworzącej nowe osobniki w populacji. Istotnym warunkiem wyboru jest to, aby wśród grona rodziców znajdowały się osobniki, które zostały najlepiej ocenione. Spośród dostępnych metod: koło ruletki, ranking, losowy dobór oraz turniej wybrano **metodę turniejową**. Z pupulacji wybiera się ilość osobników (tzw. rozmiar turnieju, która kwalifikuje się do zawodów). Następnie dwa najlepsze osobniki są wybierane jako rodzice dla nowego osobnika. Rozmiar turnieju jest stały dla każdej instancjii testowej, jednak jego dobór jest losowy. Przedmiotem badania jest również wpływ wielkości turnieju na osiągane najlepsze rozwiązanie.

Zalety:

- nie wymaga znajomości optymalizowanej funkcji, konieczny jest jedynie operator porównywania osobników
- nie wymaga żadnych informacji statystycznych opisujący populację
- jest wygodna dla wariantu z ciągłą ewolucją populacji (nie tworzy się całkiem nowej populacji n+1 na bazie populacji n, lecz część osobników z populacji wymiera i jest zastąpiona przez nowe osobniki pochodzące od operatora krzyżowania (**stały rozmiar populacji!!!**)

Operatory krzyżowania

Operator krzyżnowania może zostać rozumiany analogicznie do rozmnażania osobników. W naszej implementacji uwzględniliśmy dwie wersje możliwego tworzenia potomków:

- kombinacja losowych fragmentów genu np.:

[0, -1 ,1, 3, 2, 4, 6, 5, 0] x [0, -1, 5, 4, 6, 3, 1, 2, 0] = [0, -1 ,1, 3, 2,5, 4, 6,0]

- wspólna cześć genotypów rodziców uzupełniona o losowe geny (analogia do dziedziczenia koloru oczu po rodzicach) np.:

  [0, -1, 1, 4, 2, 3, 0, -1, **5, 6, 7**, 0] x [0, -1, 2 , 3, **5, 6, 7**, 0, -1, 4, 0] = [0, -1, **5, 6, 7**, 4, 0, -1, 3, 2, 1]

Częstotliwość wywołania operatów krzyżowania jest określana programowo, i stanowi obiekt badania, jak należy ją dobrać by działanie algorytmu było prawidłowe. Prawdopodobieństwo wywołania operatora krzyżowania pierwszego oraz drugiego typu jest równe.

Operatory mutacji

Mutacja zapewnia różnorodność osobników i jest aplikowana z pewną dozą prawdopodobieństwa na losowego osobnika populacji. Zaletą tej procedury jest spora **większa różnorodność rozwiązań,** jednak z **groźbą zepsucia dobrego rozwiązania**. W związku z charakterem algorytmu zaproponowano kilka mutacji, które zapewniają przeszukanie względnie dużej części przestrzeni rozwiązań.

- dodanie(1)/usunięcie(2) znacznika kolejnego samochodu:

  Zmienia w rozwiązaniu ilość pracujących samochodów o jeden np.:

  (1) [0, -1, …, -1, 4, 5, -1, 150, -1, … ,0] → [0, -1, …,-1, 4, **-1**, 5, 150, -1, … ,0]

  (2)[0, -1, …,-1, 4, 5, ~~-1~~, 150, -1, … , 0] → [0, -1, …, -1, 4, 5, 150, -1, … , 0]

- przesunięcie znacznika samochodu:

  Zmienia ilość odwiedzonych ulic dla dwóch sąsiednich samochodów np.:

  [0, -1, 1, 2, 3, -1, 4, 5, 6, 0] → [0, -1, 1, 2, 3, 4, 5, -1, 6, 0]

- dodanie(1)/usunięcie(2) znacznika powrotu załadowanego samochodu do sortowni

  Zmienia obiążenie samochodu oraz trasę np.:

  (1) [0, -1, 1, 2, 3, -1, 4, 5, 6, 0] → [0, -1, 1,**0**, 2, 3, -1, 4, 5, 6, 0]

  (2) [0, -1, 1, 0, 2, 3, -1, 4, 5,~~0~~, 6, 0] → [0, -1, 1, 0, 2, 3, -1, 4, 5, 6, 0]

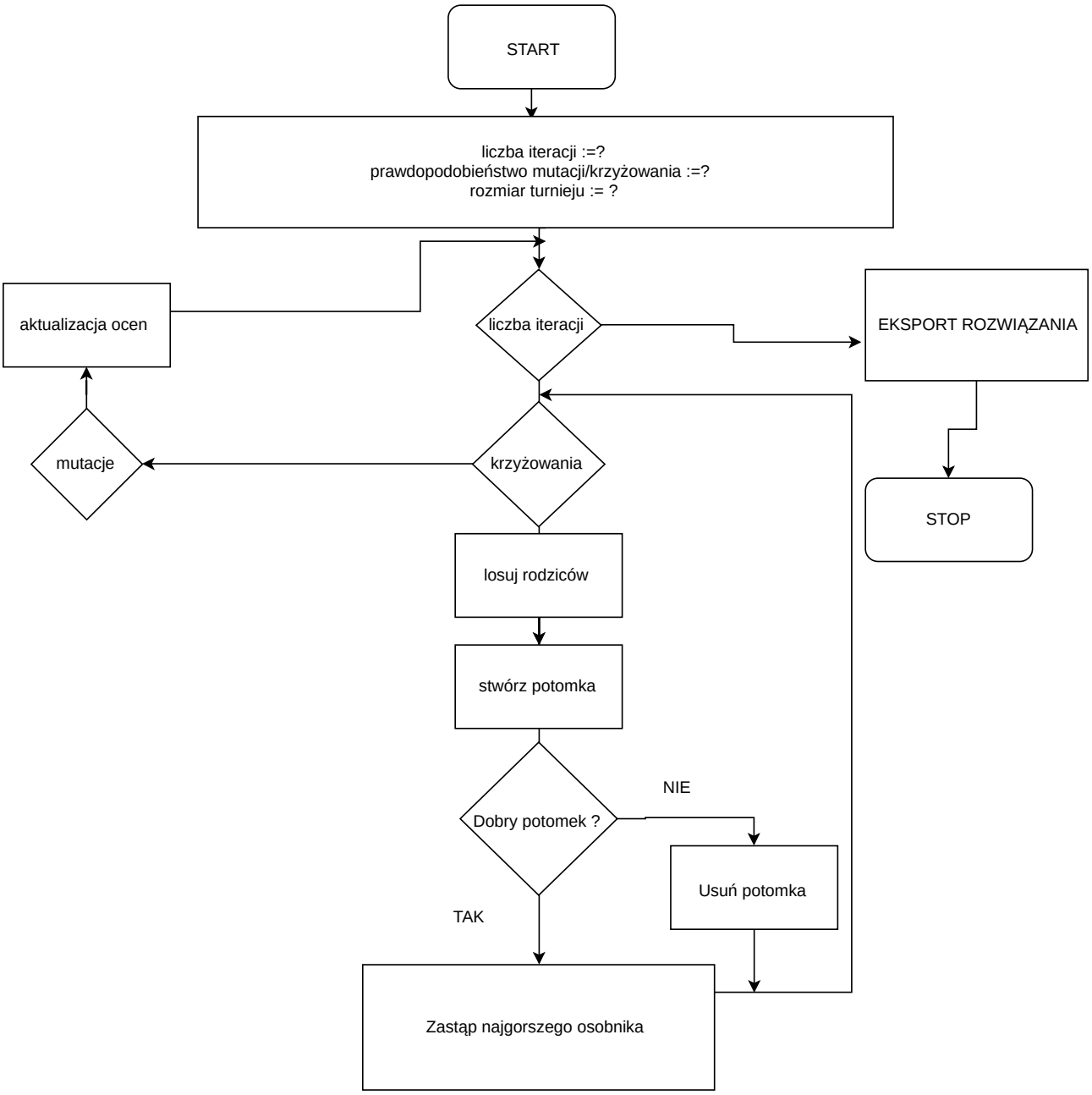
- przesunięcie znacznika posesji

  Zmiana odwiedzanych węzłów przez samochody(1) lub zmiaa kolejności w trasie (2)

  (1) [0, -1, 2, 3, 5, -1, 6, 7, **8**, 9, 10, 0] → [0, -1, 2, **8**, 3, 5, -1, 6, 7, 9, 10, 0]

  (2) [0, -1, 1, 2, 3, 4, **5, 6**, 0] → [0, -1, 1, 2, 3, 4, **6, 5**,0]

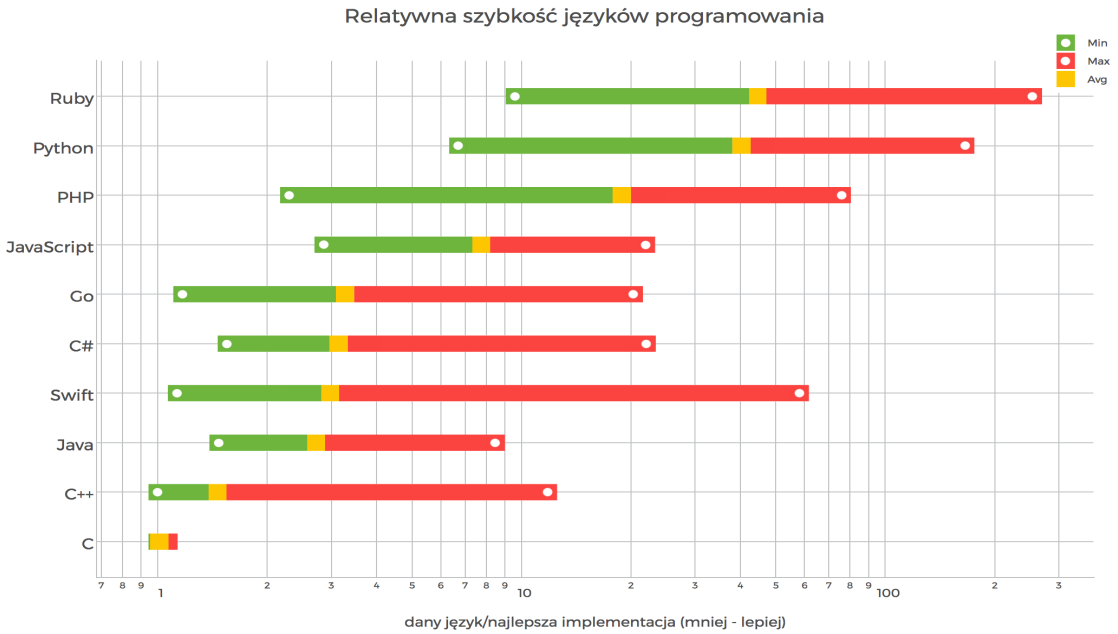
Uproszczony schemat blokowy



Wykorzystywane oprogramowanie

Python

Język programowania wysokiego poziomu, używany przez w wersji 3.6. Został wybrany poprzez bardzo szeroki pakiet bibliotek standardowych oraz funkcjonalnych (patrz. następny punk). Licencja PSFL (jest absolutnie darmowy, nawet w zastosowaniach komercyjnych). Jednak cenę, którą płacimy za wygodę i dostępność wielu komponentów jest relatywna szybkość (a raczej wolność) w stosunku do innych języków.



Rys 1: Porównianie szybkości 10 naj. popularnych języków programowania. (źródło:https://bulldogjob.pl/ckeditor_assets/pictures/181/content_szybkosc_jezyki_programowania_bulldogjob.png)

Biblioteki

NumPy

Biblioteka wspomagająca obliczenia dużych, wielowymiarowych tablic, macierzy - współpracująca m. in z językiem Python. Dostarczająca gotowe implementacje procedur psedolosowego doboru elementów wektorów, kombinacji liniowej, permutacji, obliczenia ilości wystąpień elementów w ciągu liczb. Umożliwia również zapis obiektów NumPy do pliku binarnego z możliwością natychmiastowego wczytania i przypisania do zmiennej w programie. Posiada szeroko zaimplementowany mechanizm wyjątków, dzięki którym ewentualne błędy mogą zostać wyłapane i poprawione. Ma bardzo szerokie zastosowania i jest bardzo często używana w związku z tym istnieje **BARDZO WIELE** materiałów, turtorialów oraz bardzo szczegółowa dokumentacja.

Matplotlib

Pakiet implementujący generację wykresów (wykresy punktowe, kołowe, histogramy, trójwymiarowe) z możliwością zapisu w formatach wektorowych z wysoką rozdzielczością. Wykorzystywany w pracy do prezentacji obliczeń oraz wizualizacji tras samochodów.

GeoPy

Klijent serwisów Geocodingowych. Dzięki połączeniu z globalną siecią Internet pozwala na znalezenie współrzędnych dla zadanego miejsca(adresu, gminy, kraju). Udostępnia również szereg zaimplementowanych funkcji do obliczania odległości między dwoma punktami na powierzchni ziemi. Powyższe obliczenia można wykonywać używając różnych modeli obliczeniowych (sferyczny, eliptyczny wraz z różnymi ich wersjami), niestety brak implementacji uwzględniającej jedynie drogi dla samochodów.

Inne

- GitLab + GIT
- HTML (eksport raportu dla wygodnego podglądu)
- PyCharm Proffesional IDE
- Pandas (przetwarzanie danych wejściowych z SQL)

Testy

Problem najkrótszej ścieżki bez załadunku

Pierwszym etapem testów, było sprawdzenie, jak zaimplementowany algorytm poradzi sobie w rozwiązywaniu postawionego problemu po nałożeniu ograniczeń upraszczających metodykę obliczania funkcji celu. Przyjęto brak wpływu ładowności aut oraz wybór maksymalnie jednego samochodu, co sprowadziło zagadnienie to Problemu Komiwojażera (symetrycznego, choć metody obliczania kosztów, poprawnie działa również dla problemu niesymetrycznego) z określonym (stałym) wierzchołkiem startowym i końcowym. Celem zabiegu było sprawdzenie jak algorytm radzi sobie z uproszczonym problemem, który jest łatwiejszy do analitycznej weryfikacji działania. Wyniki działania algorytmu, można również porównać z dostępnymi solverami problemu komiwojażera w sieci oraz dla małych rozmiarów przeprowadzić przegląd zupełny rozwiązań.

Manualne testy

Sprawdzanie odpowiedzi algorytmu na zmiany metod obliczania wartości funkcji celu oraz reakcji wyniku algorytmu na nie (podrożenie kosztu aktywacji pojazdu tysiąc-krotne, zmniejszenie kary za przeładowanie, itp…). Wynik wpływu zmienianych parametrów był łatwy do przewidzenia i pozwalał odpowiedzieć na fundamentalne pytanie

TDD

Test-driven development tworzenie oprogramowania polegające na powtarzaniu w stosunku do każdej nowej funkcjonalności kroków :

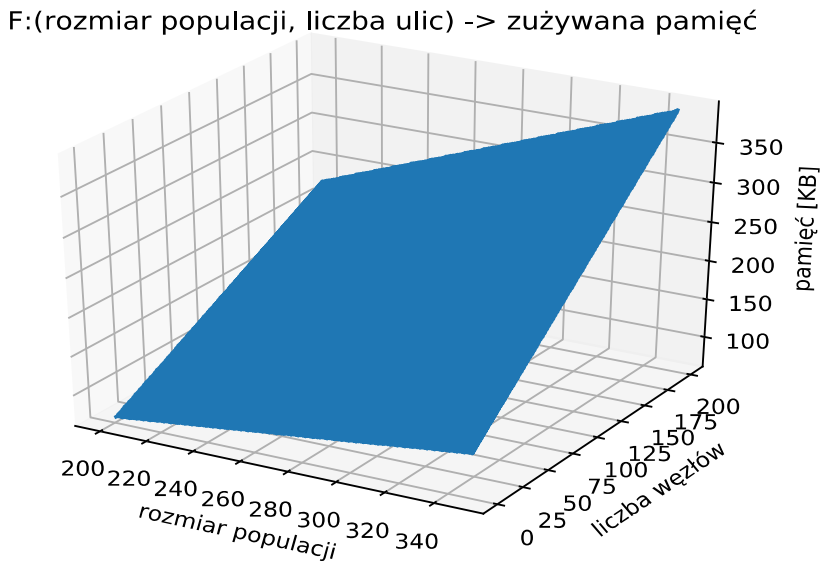
1. Implementacja automatycznego testu (zastosowano Python unittest)
2. Niepowodzenie testu
3. Implementacja funkcjonalności kodu
4. Ponowne uruchomienie testu z 100% skutecznością

Przybliżona złożoność algorytmu

Złożoność algorytmu oraz jego operatorów jest przydatną informacją w przypadku podawania na wejścieproblemów dużego rozmiaru. Obecny dostęp do dużych rozmiarów pamięci oraz szybkich procesorów nie wymaga najniższej możliwej złożoności obliczeniowej oraz oszczędności operacji jednak w przypadku testowania funkcji, czy implementacji bardziej złożonych struktór bez mechanizmu mockowania bywa to problematyczne i czasochłonne.

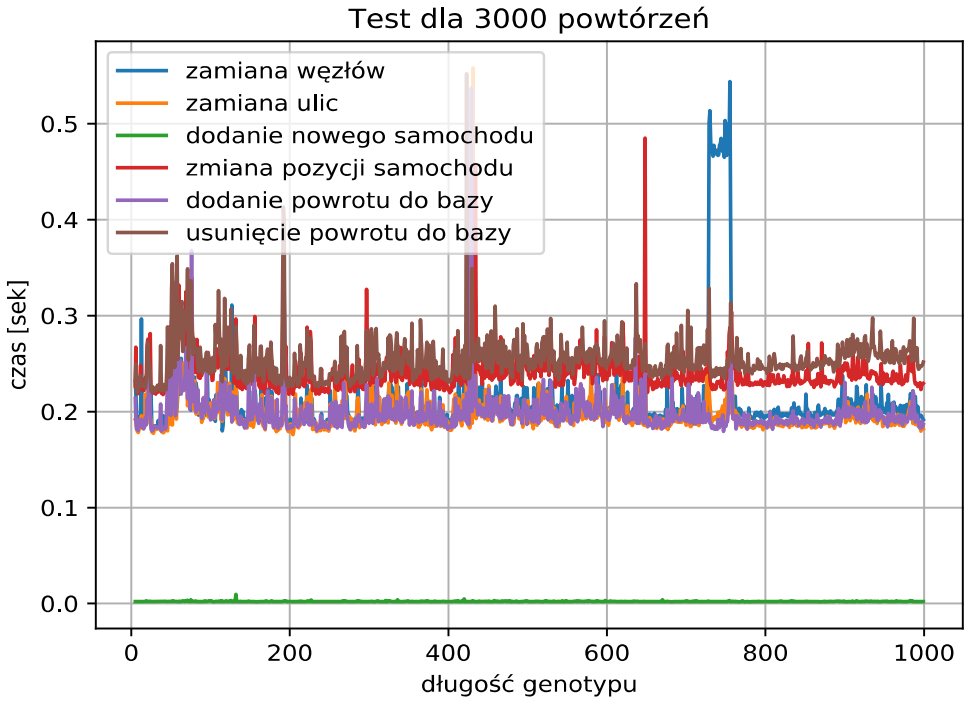
Złożność pamięciowa populacji

Własnością ewolucji populacji „w miejscu” jest brak konieczności alokacji pamięci na nową przyszłą populację. Cała pamięc potrzebna do poprawnego działania algorytmu jest zużywana tylko do przechowywania stałej ilości osobników (o różnej długości wektora decyzyjnego) oraz ewentualnie tworzonych obiektów tymczasowych, które są usuwane przez Python Gc(*ang. garbage collector*). Zatem bez konieczności stosowania zaawansowanych narzędzi profilerskich, wykorzystując bibliotekę *Pympler* dokonano pomiaru zajmowanej pamięci przez obiekt klasy Populacja,dla różnych instancji. Należy jednak zaznaczyć, że jest to obraz barzdo przybliżony, aby uzyskać wykres zajmowanej pamięci dynamicznie podczas działania algorytmu, należało by przynajmniej po każdej iteracji zapisywać „wagę” obiektu Populacji i wykreślić jej kształt. Określanie złożoności pamięciowej, z dokładnością do chwilowych obiektów dało by bardziej szczegółowy punkt do analizy, jednak nie jest to naszym celem.

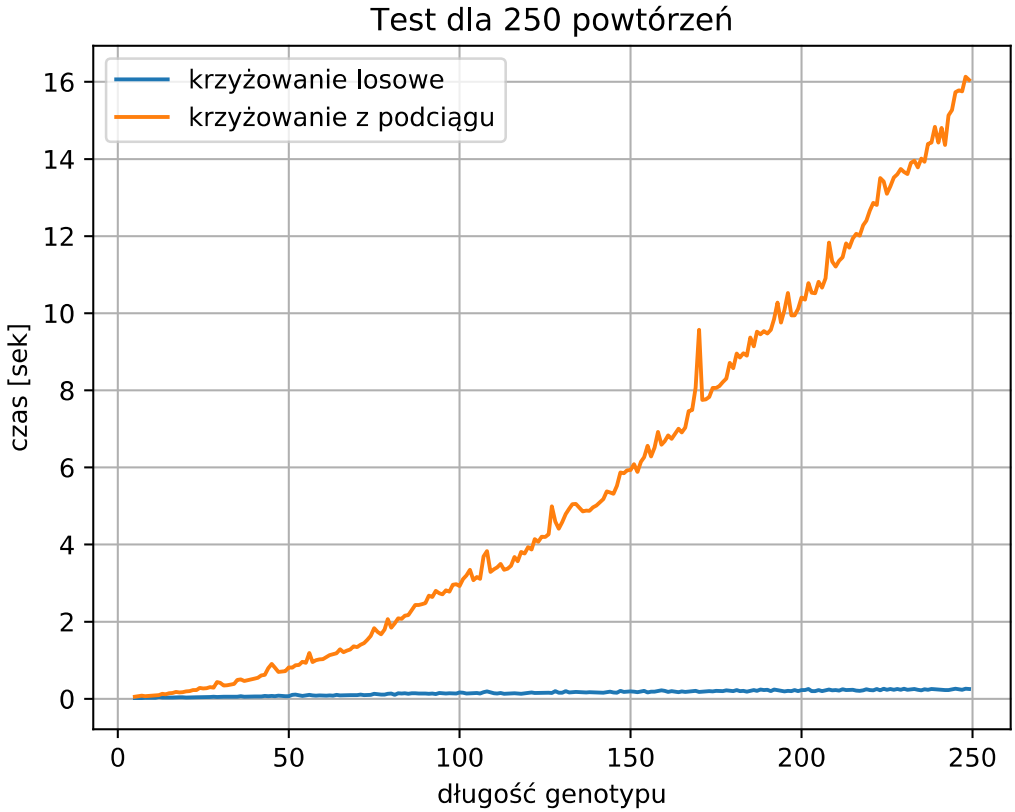


Rys 2: Prezentacja złożoności pamięciowej populacji

Złożoność czasowa operatorów



Rys 3: Prezentacja złożoności operatorów mutacji



Rys 4: Prezentacja złożoności operatorów krzyżowania

Otrzymane wyniki

Problem TSP

- Dane wejściowe

Tab. 1: Macierz kosztów dla TSP

x	105.91	211.81	317.71	423.61	529.51	635.40
105.91	x	105.91	211.81	317.71	423.61	529.51
211.81	105.91	x	105.91	211.81	317.71	423.61
317.71	211.81	105.91	x	105.91	211.81	317.71
423.61	317.71	211.81	105.91	x	105.91	211.81
529.51	423.61	317.71	211.81	105.91	x	105.91
635.40	529.51	423.61	317.71	211.81	105.91	x

- Raport z testu

Tab 2: Parametry wejściowe algorytmu i odpowiedź

16	6	12	13	1694.48
16	**9**	**1**	**5**	**1482.67**
20	**11**	**11**	**8**	**1482.67**
19	2	4	8	2118.09
6	1	13	7	2118.09
9	4	9	6	1694.48
			średnia	1887.03
			odchylenie standardowe	395.74
			współczynnik zmienności	20.97%

Tab 3: Udział operatorów krzyżowania

33.33%	66.67%	**1482.67**
42.86%	57.14%	2118.08
37.50%	62.50%	1694.48
42.86%	57.14%	1694.48
25.00%	75.00%	**1482.67**
62.50%	37.50%	**1482.67**
66.67%	33.33%	2118.09
50.00%	50.00%	2118.09
33.33%	66.67%	1694.48

- Analiza wyników

  Otrzymanie najlepsze rozwiązanie, za pomocą algorytmu to:

$$0 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow 5 \rightarrow 1 \rightarrow 0$$

```
for i in range(100000):
    x.street_order = np.arange(1, 7)
    np.random.shuffle(x.street_order)
    x.calculate_cost_value(inp_data)
    if x.cost_function_value < best_solution:
      best_solution = x.cost_function_value
```

Text 1: Uproszczony skrypt przeglądu zupełnego dla problemu

  Otrzymanie najlepsze rozwiązanie, za pomocą przeglądu zupełnego:

$$0 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow 5 \rightarrow 1 \rightarrow 0$$

Rozważany problem TSP z ustalonym węzłem początkowym oraz końcowym został rozwiązany przez nasz algorytm w pełni (znalezione zostało rozwiązanie optymalne). Liczba możliwych tras była równa 6! = 720. Przedstawiona instancja testowa miała na celu sprawdzenie czy algorytm przeszukuje całą przestrzeń rozwiązań(w odpowiednio dobranie dużym czasie), przez sprawdzenie czy w którejś iteracji pojawi się zadana losowo wariancja; oraz czy rozwiązania najlepsze (optymalne) zostaną znalezione.

Skuteczność algorytmu została przebadana dla złożoności nastaw nie gorszych niż przegląd zupełny. Miało to na celu sprawdzenie czy algorytm jest bardziej wydajny obliczeniowo w rozwiązywaniu problemu niż najgorsza (czasowo)- metod siłowa. Najlepsze rowiązanie zostało znalezione (w jednej z instancji testowych) już po przeglądzie **200 z 720** możliwych kombinacji- przy odpowiednio niskim współczynniku rozmiaru turnieju.

Problem właściwy

Instancja I (Gorzyce Small 2)

- Dane wejściowe

Tab 4: Zestawienie dostępnych samochodów

nazwa	pojemnosc	koszt aktywacji	koszt/1km
Śmieciarka 0	90.00	200.00	1.00
Śmieciarka 1	100.00	250.00	2.00
Śmieciarka 2	400.00	400.00	2.50
Śmieciarka 3	200.00	300.00	2.30
Śmieciarka 4	350.00	350.00	3.00
Śmieciarka 5	350.00	340.00	2.00

Tab 5: Wykaz szacowanej ilości koszy oraz lokalizacji

miejscowość	ulica	numer	ilość koszy	Ilość posesji	latitude	longitude
Wodzisław Śl.	1 maja	16	0	0	49.97	18.54
Rogów	Czyżowicka	9	36	18	49.99	18.35
Rogów	Dębowa	14	10	5	50.00	18.36
Rogów	Klonowa	10	10	5	49.99	18.37
Rogów	Krótka	1	4	2	49.99	18.35
Rogów	Krzywa	4	54	27	49.99	18.35
Rogów	Leśna	11	102	51	49.99	18.36
Rogów	Lipowa	7	42	21	49.98	18.36
Rogów	Palarni	3	20	10	49.99	18.35
Rogów	Parkowa	4	28	14	49.99	18.35
Turza Śl.	27 Marca	20	106	53	49.97	18.45
Turza Śl.	Bogumińska	40	228	114	49.97	18.44
Turza Śl.	Dalków	18	100	50	49.98	18.44
Turza Śl.	Graniczna	25	146	73	49.96	18.46
Turza Śl.	Kolejowa	10	78	39	49.97	18.43
Turza Śl.	Kosciuszki	35	202	101	49.97	18.44
Turza Śl.	Ligonia	14	177	59	49.97	18.46
Turza Śl.	Mszańska	18	184	92	49.98	18.47
Turza Śl.	Piaski	6	18	6	49.98	18.46

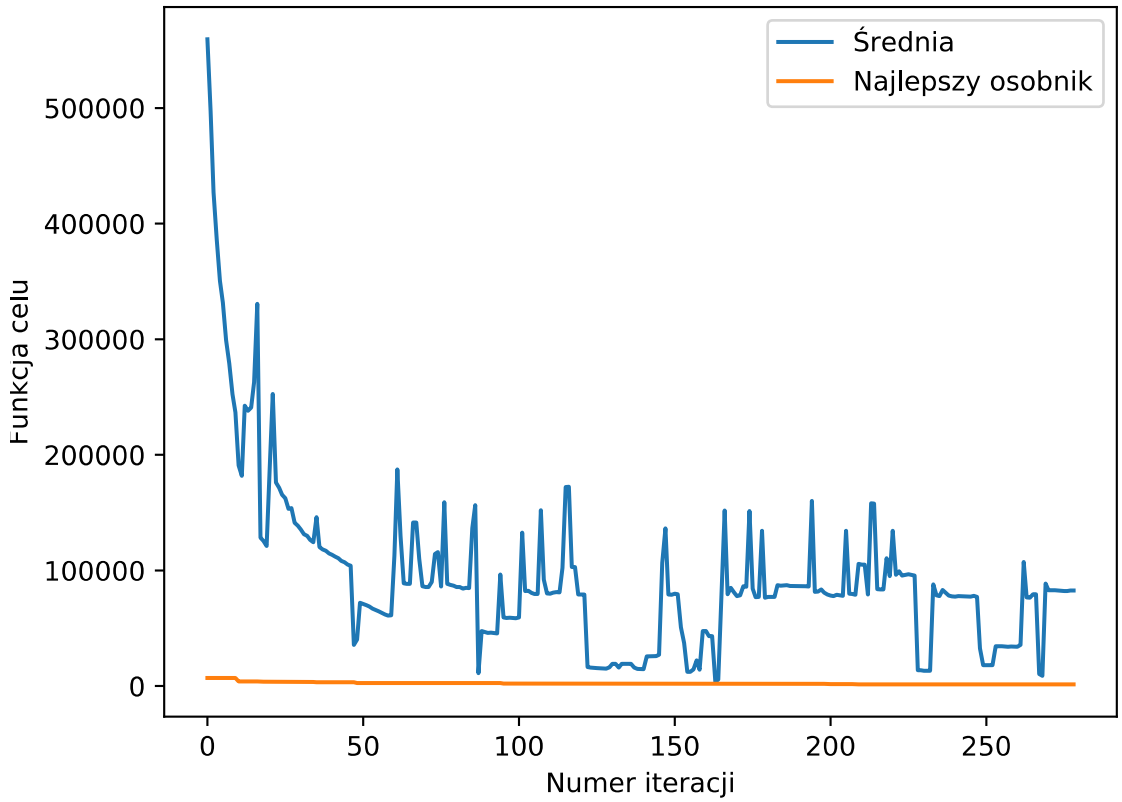
- Raport z testów

Table 6: Zestawienie parametrów wejściowych i wyniku dla różnych populacji startowych

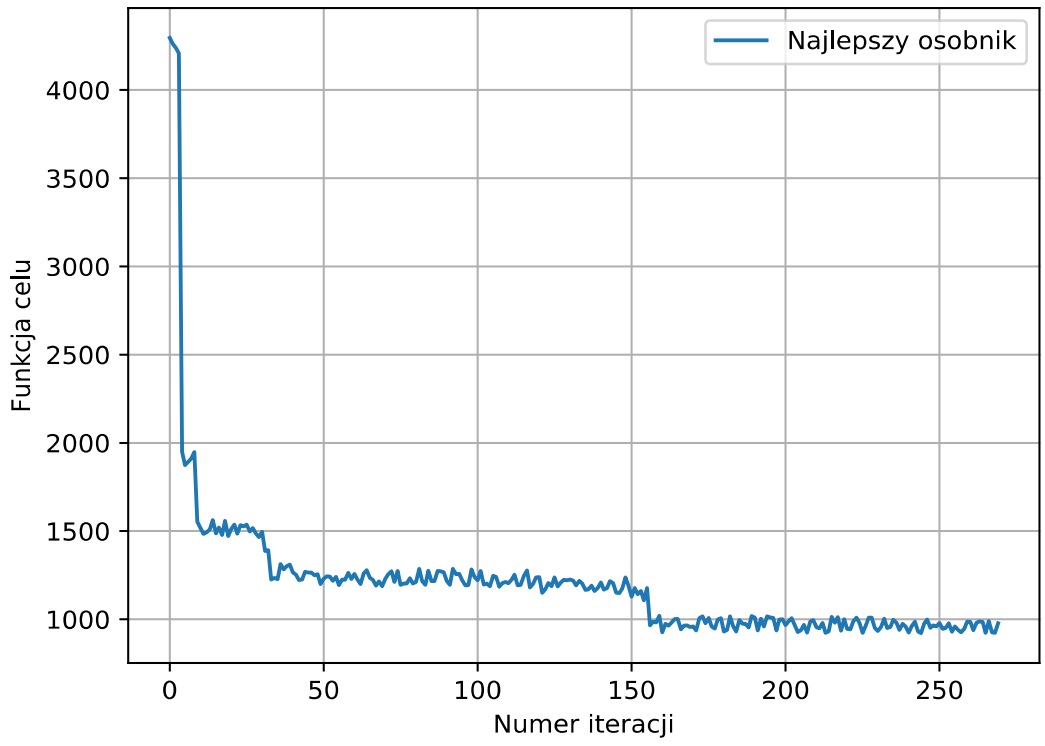
Populacja	Liczba iteracji	P(krzyżowanie)	P(mutacja)	Rozmiar Turnieju	Funkcja celu
212	297	13	13	13	904.31
232	285	7	4	11	983.59
153	129	5	14	14	997.65
222	200	11	5	11	1013.48
162	249	8	12	6	1018.38
237	85	7	11	7	1101.67
200	264	12	6	3	1127.78
196	180	6	6	13	1197.18
248	90	3	14	2	1938.95
				średnia	1142.55
				odchylenie standardowe	311.09
				współczynnik zmienności	27.23%

Table 7: Zestawienie wyniku testu dla stałej populacji początkowej oraz parametrów

124.00	338.00	12.00	13.00	7.00	984.91
124.00	338.00	12.00	13.00	7.00	999.54
124.00	338.00	12.00	13.00	7.00	1007.47
124.00	338.00	12.00	13.00	7.00	1020.76
124.00	338.00	12.00	13.00	7.00	1052.37
124.00	338.00	12.00	13.00	7.00	1128.88
				średnia	1006.57
				odchylenie standardowe	56.72
				współczynnik zmienność	5.63%



Rys 5: Porównanie wartości średniej oraz najlepszej



Rys 6: Wartość oceny najlepszego osobnikai

Instancja II (Gorzyce Small)

Tab 8: Zestawienie parametrów wejściowych i wyniku dla różnych populacji startowych

Populacja	Liczba iteracji	P(krzyżowanie)	P(mutacja)	Rozmiar Turnieju	Funkcja celu
214.00	256.00	14.00	11.00	28.00	3058.45
168.00	225.00	11.00	16.00	20.00	3119.03
218.00	346.00	6.00	7.00	32.00	3146.68
294.00	281.00	9.00	12.00	29.00	3168.58
277.00	320.00	12.00	16.00	24.00	3186.32
191.00	179.00	19.00	14.00	18.00	3300.93
172.00	307.00	12.00	8.00	20.00	3349.68
184.00	181.00	6.00	11.00	16.00	3856.88
300.00	74.00	12.00	12.00	24.00	4021.41
239.00	146.00	16.00	6.00	10.00	4160.83
279.00	26.00	20.00	15.00	11.00	4270.28
186.00	144.00	13.00	19.00	9.00	4984.91
154.00	50.00	13.00	19.00	18.00	5653.11
242.00	121.00	16.00	6.00	7.00	6914.20
				średnia	4013.66
				odchylenie stadrardowe	1142.48
				współczynnik zmienności	28.46%

Tab 9: Porównanie skuteczności operatorów krzyżowania

Krzyżowanie losowe	33	22	23	66	56	60	62	63	67	33	27	30	32	1
Krzyżowanie podciąg	2	2	2	4	4	5	8	5	5	3	2	2	3	1

Optymalny dobór parametrów dla instancjii I

W celu dobrania optymalnej ilości modyfikacji (krzyżowań i mutacji) na jedną iterację, przeprowadzono test, w którym dla jednakowej ilości operacji (czasie liczenia) zmieniano stosunek ilości iteracji do prawdopodobieństwa mutacji i krzyżowań, wyniki, pogrupowane, przedstawia tabela 10.

Tab 10: Wpływ wartości prawdopodobieństwa mutacji na wartość funkcji celu

P(krzyżowanie)	średnia Funkcji celu
10	3405.12
5	3411.63
3	3443.09
1	3504.44
2	3507.97

Powyższe wyniki okazały się być mało wiarygodne, gdyż najlepsze rozwiązania były osiągane dla niskiego prawdopodobieństwa mutacji (czyli wąskiego obaszru przeszukiwań) jednak okazało się, że były one także najgorsze, wysokie prawdopodobieństwo było bardziej skupione wokół pewnej wartości, dlatego przeprowadziliśmy testy na większej ilości instancji (ponad 15k).

Tab 11: Wpływ wartości prawdopodobieństwa mutacji na wartość funkcji celu II

Liczba powtórzeń	P(krzyżowanie)	Liczba iteracji	średnia	minimum	maximum	odchylenie standardowe
1215	18	170	3114	2883	3772	95
1215	33	90	3122	2911	3999	104
2340	7	410	3107	2833	4873	117
1215	23	130	3111	2873	4968	105
1215	14	210	3124	2890	4980	110
1134	6	490	3133	2883	5069	143
1170	8	370	3121	2881	5080	140
1200	9	350	3170	2928	5267	252
1243	10	300	3160	2694	5313	252
1267	60	50	3154	2875	5342	256

W ten sposób ustalono, że podczas 1 iteracji dokonywane są mutacje na 36% populacji oraz średnie wykorzystanie operatorów:

Tab 12: Udział poszczególnych operatorów w tworzeniu rozwiązania

Krzyżowanie losowe	Krzyżowanie podciąg	usuń samochód	przesuń samochód	dodaj powrót	usuń powrót	dodaj auto	usuń auto	zamień składowe
109,28	0,053	0	0	0,73	1,04	0	0,95	0,91

Wynika z powyższego, że niektóre operatory nie przyczyniają się do polepszania rozwiązania,dlatego z nich zrezygnowano. Stwierdzono, że mutacje najbardziej pomagają w końcowychiteracjach, dlatego ustalono ich dominującą rolę w końcowych 20% iteracjii.

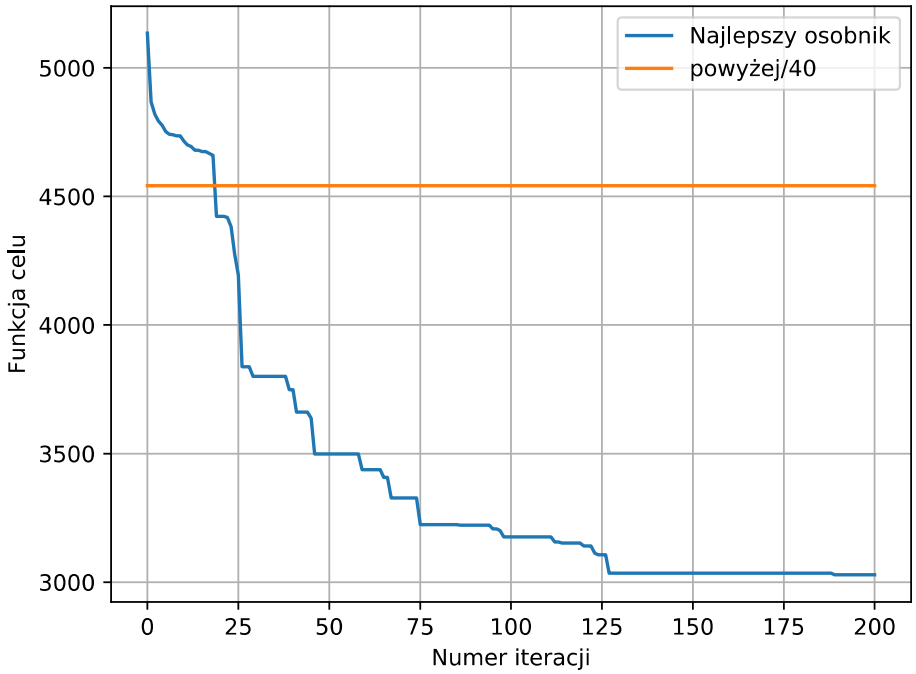
Optymalne parametry (dla Instancjii I):

- P(krzyżowanie) 25%

  losowe: 98%

  podciąg: 2%
- P(mutacja) 11%
- Rozmiar turnieju: 35% populacji
- Rozmiar populacji 250

Przy takich ustawieniach, dla 200 iteracji (około 18k obliczeń Fcelu): 3028



Rys 7: Funkcja celu dla optymalnego doboru parametrów- pełen problem

Wynik:

auto 2

smieciarka2, poj: 500

kurs 0

1.36--Turza Śl., Tysiąclecia, 135
2.29--Turza Śl., Ligonia, 177
3.38--Turza Śl., Wodzisławska, 148
4.35--Turza Śl., Świerkowa, 8
5.RAZEM: 468 / 500 koszy

kurs 1

1.28--Turza Śl., Kosciuszki, 202
2.18--Rogów, Wiejska, 54
3.1--Rogów, Czyżowicka, 36
4.16--Rogów, Szkolna, 3
5.9--Rogów, Parkowa, 28
6.12--Rogów, Powstańców, 14
7.7--Rogów, Lipowa, 42
8.2--Rogów, Dębowa, 10

cd. **kurs 1**
9.17--Rogów, Słoneczna, 2
10.20--Rogów, Wrzosowa, 6
11.5--Rogów, Krzywa, 54
12.3--Rogów, Klonowa, 10
13.RAZEM: 461 / 500 koszy

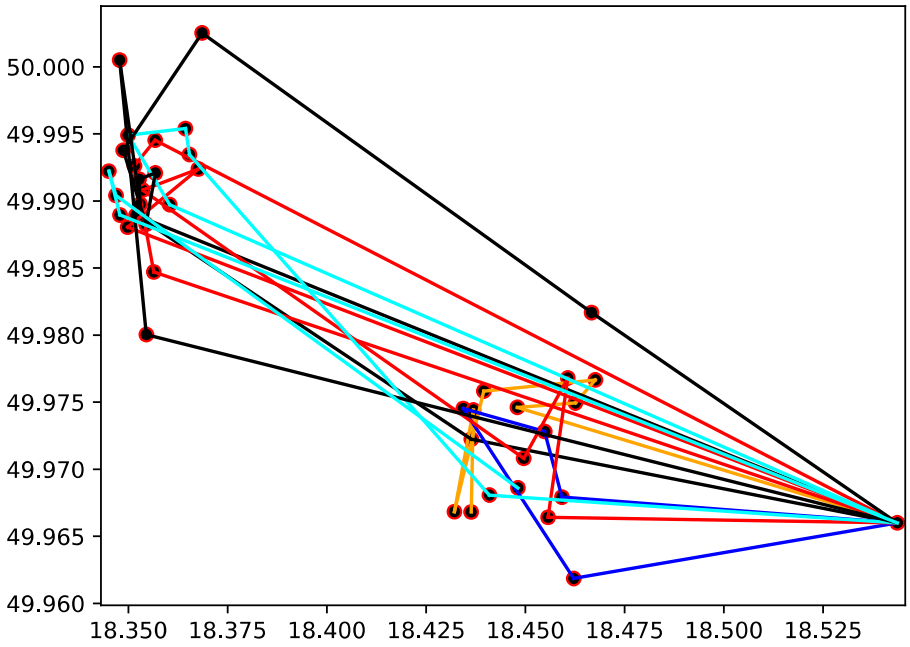
kurs 2

1.10--Rogów, Polna, 98
2.22--Rogów, Wyzwolenia, 370
3.RAZEM: 468 / 500 koszy

kurs 3

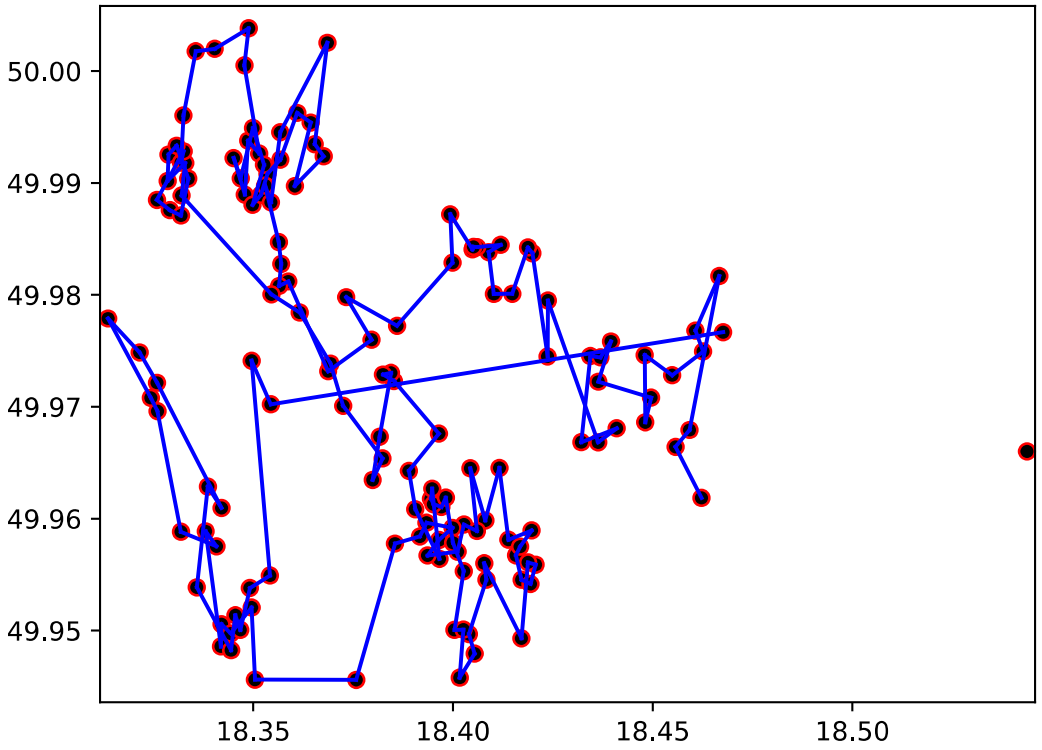
1.39--Turza Śl., Wspólna, 30
2.31--Turza Śl., Piaski, 18
3.33--Turza Śl., Powstańców, 208
4.37--Turza Śl., Korfantego, 34
5.19--Rogów, Wodzisławska, 60
6.25--Turza Śl., Dalków, 100
7.RAZEM: 450 / 500 koszy

Samochody nie są przepełnione, wybrane zostało najbardziej pojemne auto (przy stosunkowo dobrej cenie). Trasy poszczególnych kursów są logicznie dobrane. Brak jest dziwnych pętli, powtórzeń itp.



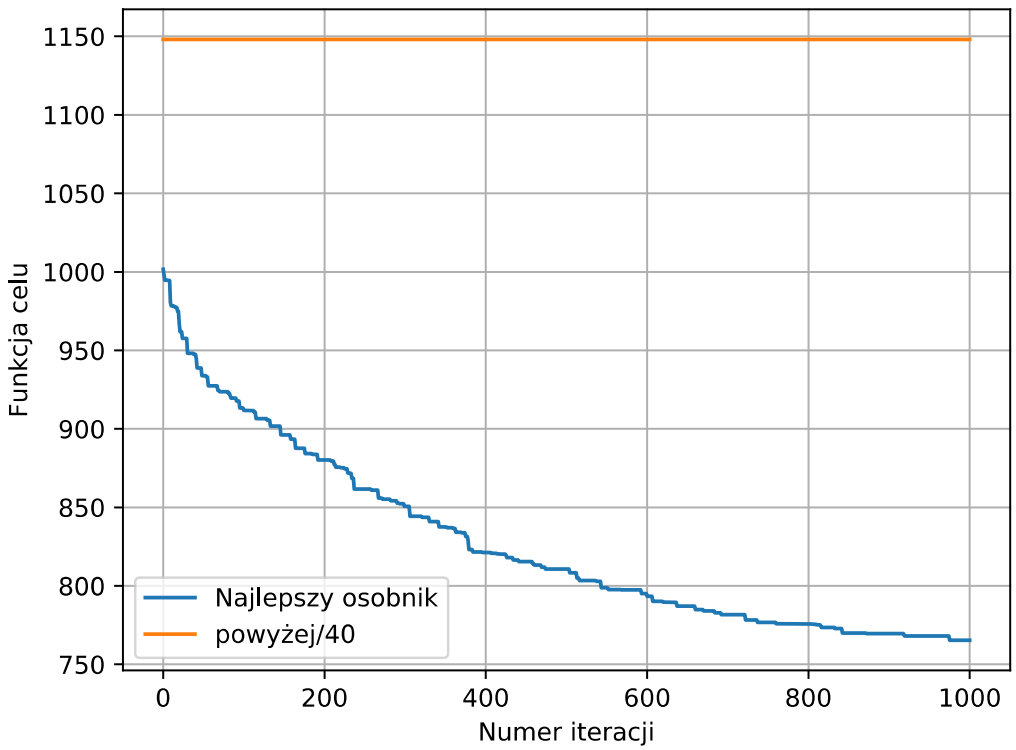
Rys 8: Wizualizacja tras przejazdów dla pełnego problemu

Dla całej gminy ciężko zweryfikować poprawność rozwiązania, jednak dla małej ilości iteracji intuicyjnie wydaje się być nieoptymalne, dopiero dla 10 000 iteracji otrzymano rozwiązanie, które można uznać za nie-najgorsze. Upraszczając problem do komiwojażera z 150 punktami, szybko można znaleźć dobre rozwiązania (nie tylko mniej iteracji, ale także czas obliczania funkcji celu jest zdecydowanie krótszy), wysuwa się myśl, że algorytm jest poprawny, tylko problem śmieciarek jest trudny.



Rys 9: Wizualizacja trasy dla problemu komiwojażera z optymalnymi parametrami

Odizolowany punkt nieobjęty trasą to baza (wyłączony z trasy).



Rys 10: Funkcja celu dla problemu komiwojażera z optymalnymi parametrami

Instancja III (Gorzyce)

Tab 13: Zestawienie parametrów wejściowych i wyniku dla różnych populacji startowych

Populacja	Liczba iteracji	P(krzyżowanie)	P(mutacja)	Rozmiar Turnieju	Funkcja celu
333.00	1110.00	16.00	8.00	26.00	20161.06
267.00	1433.00	15.00	12.00	20.00	20676.18
338.00	809.00	5.00	18.00	34.00	22423.67
300.00	480.00	18.00	9.00	38.00	22954.42
303.00	881.00	5.00	11.00	25.00	24097.38
324.00	268.00	8.00	20.00	46.00	28992.07
270.00	209.00	13.00	19.00	43.00	31374.52
				średnia	24382.76
				odchylenie standardowe	4235.79
				współczynnik zmienności	17.37%

Tab 14: Porównianie efektywności operatorów krzyżowania

Krzyżowanie losowe	Krzyżowanie podciąg
161	1
92	1
251	5
82	1
251	4
123	1
107	2

Wnioski i ocena algorytmu

- Algorytm doskonale radzi sobie z problemem komiwojażera, znajduje stosunkowo dobre rozwiązania w krótkim czasie
- Dla problemu śmieciarek także znajduje rozwiązanie optymalne wg funkcji celu. Konsultując parametry funkcji celu według potrzeb można próbować zastosować program komercyjnie
- Przeprowadzone testy pokazały, że tak naprawdę najbardziej wykorzystywany jest operator krzyżowania losowego, warto by pomyśleć nad dodatkowymi operatorami, które mogły by dawać lepsze rozwiązania i były by efektywniej używane
- Wprowadzenie do rozwiązania początkowego kilku alfabetycznie posortowanych (wg miejscowości) ciągów, (ulice w tej samej miejscowości są raczej blisko siebie) zauważalnie poprawiło działanie algorytmu, pojawił się pomysł, aby rozwiązania początkowe mieszać, ale tylko w obrębie danych ulic, a potem całych miejscowości, i do tego dodać z 30% losowo generowanych ciągów, tak aby za bardzo nie zawężać przestrzeni
- Algorytm jest uniwersalny, przez co, zmieniając tylko funkcję celu można go używać do optymalizowania innych zagadnień
- Najdłużej (w jednej iteracji) trwa obliczanie funkcji celu, optymalizacja w tym zakresie może przynieść znaczne przyspieszenie działania