Projekt z Badań operacyjnych

System optymalizacji transportu przez granicę

Skład zespołu:
Katarzyna Błachut
Jan Gołda
Piotr Ćwiklicki

Opiekun:
dr hab. inż. Joanna Kwiecień

Informatyka, WIEiT 2019/2020

1. Cel oraz założenia projektu

Celem projektu było znalezienie rozwiązania problemu przewiezienia danych towarów w danej ilości przez granicę, ponosząc przy tym jak najmniejszy koszt. Pod uwagę brane były cła na poszczególnych przejściach dla każdego typu towaru, pojemność i paliwo ciężarówek, koszt paliwa oraz odległości niezbędne do pokonania dla każdego przejścia (dokładny opis w rozdziale 2).
W celu znalezienia optymalnego rozwiązania problemu posłużono się technikami badań operacyjnych. Do wygenerowania rozwiązań wykorzystany został algorytm oparty na algorytmie pszczelim.

2. Opis zagadnienia

2.1. Sformułowanie problemu

Za pomocą z góry ustalonej ilości ciężarówek, o ograniczonej ładowności, należy przewieźć zadaną ilość towaru różnych typów przez granicę. Do dyspozycji mamy różne przejścia graniczne, jednak różnią się one odległością w jakiej się znajdują oraz wysokością cła na poszczególne typy towarów.
Celem jest przewiezienie wszystkich towarów przy jak najmniejszym koszcie, co wymaga takiego przydziału towarów do ciężarówek oraz wyboru przejść granicznych aby koszty za paliwo oraz za cło były jak najmniejsze.

2.2. Zastosowania

Rozwiązanie tego problemu można wykorzystać w sytuacji przewozu towarów przy odpowiednio dużej flocie ciężarówek przez jedną granicę, ale także w sytuacji gdy towary trzeba przewieźć przez kilka państw - ten przypadek jest najbliższy rozwiązywanemu problemowi, ponieważ w świecie rzeczywistym cła są raczej ustalane dla całego kraju, więc jedyny przypadek, kiedy koszty na poszczególnych przejściach będą się różnić występuje w sytuacji przekraczania wielu granic, ponadto przy tak długiej drodze koszty paliwa mają już większe znaczenie.
Alternatywnie rozwiązanie można wykorzystać do zaplanowania jak najskuteczniejszego przemytu towarów (np. mięsa do Szwajcarii) - wówczas koszt danego towaru na danym przejściu będzie wyrażał jak dobrze dane przejście jest pilnowane (prawdopodobieństwo złapania przez celnika).

2.3. Możliwe rozszerzenia problemu

Z racji na ograniczenia czasowe, w projekcie uwzględniliśmy ograniczony problem, eliminując niektóre z początkowych pomysłów o które w przyszłości można by go rozszerzyć. Pominięte pomysły to:

- Zakazane kombinacje towarów w jednej ciężarówce
- Objętość towarów i dodatkowe ograniczenie pojemności ciężarówek
- Heterogeniczne ładowności ciężarówek
- Termin przydatności towarów i czas podróży przez dane przejścia

2.4. Model matematyczny

Szukanie rozwiązania problemu musimy zacząć od dokładnego jego sformułowania w sposób matematyczny.

2.4.1. Dane

$n \in \mathbb{N}$ - ilość przejść granicznych
$m \in \mathbb{N}$ - ilość typów towarów
$p \in \mathbb{N}$ - ilość ciężarówek
$v \in \mathbb{R}_+$ - ładowność jednej ciężarówki
$f \in \mathbb{R}_+$ - koszt paliwa za jednostkę dystansu
$c_{ij} \in \mathbb{R}$ - cło za przewiezienie jednej jednostki j-tego towaru przez i-te przejście graniczne
$d_i \in \mathbb{R}_+$ - odległość trasy w jednostkach dystansu przez i-te przejście
$t_j \in \mathbb{R}_+$ - ilość danego towaru do przewiezienia przez granicę

$i \in [0, n); j \in [0, m); k \in [0, p); i,j,k \in \mathbb{N}$

2.4.2. Szukane

$a_k \in \mathbb{N}$ - przez które przejście będzie jechać k-ta ciężarówka
$b_{kj} \in \mathbb{R}_{+0}$ - ilość j-tego towaru do przewiezienia w k-tej ciężarówce

2.4.3. Ograniczenia

1. Ograniczenie ładowności ciężarówek ⇔ ciężarówka nie może wziąć więcej niż wynosi jej ładowność

$$\forall_k \sum_j b_{kj} \leq v$$

2. Ograniczenie ilości towaru pomiędzy ciężarówkami ⇔ trzeba przewieźć cały towar

$$\forall_j \sum_k b_{kj} = t_j$$

2.4.4. Minimalizowana funkcja

Suma po każdej ciężarówce z sumy ceł za towary w tej ciężarówce oraz kosztu paliwa.

$$f(a,b) = \sum_k^p (\sum_j^m (b_{kj} * c_{a_k,j}) + f * d_{a_k})$$

3. Opis algorytmu

3.1. Idea i algorytm podstawowy

W celu rozwiązania postawionego problemu zastosowano algorytm pszczeli. W podstawowej wersji algorytm ten polega na wylosowaniu początkowej puli rozwiązań początkowych, a następnie wyznaczenia jakości każdego rozwiązania. Następnie na podstawie parametru jakości wybierane są rozwiązania dobre oraz elitarne. W dalszej części tylko one brane są pod uwagę. Dla każdego dobrego lub elitarnego rozwiązania generowane są następne rozwiązania z jego bliskiego otoczenia (które musi zostać zdefiniowane dla danego problemu), ilość generowanych rozwiązań z otoczenia rozwiązania elitarnego i otoczenia rozwiązania dobrego są określone dwoma parametrami (jeden dla dobrych i jeden dla elitarnych). Następnie spośród otoczenia wybierane jest najlepsze rozwiązanie dobre i najlepsze rozwiązanie elitarne. Do tych rozwiązań dokładane są kolejne rozwiązania wygenerowane losowo i cała procedura jest powtarzana do momentu spełnienia kryterium stopu lub wykonania żądanej ilości iteracji.

Pseudokod z The Bees Algorithm Technical Note [1]:

1. Initialise population with random solutions.
2. Evaluate fitness of the population.
3. While (stopping criteria not met) //Forming new population.
    a. Select elite bees.
    b. Select sites for neighbourhood search.
    c. Recruit bees around selected sites and evaluate fitness.
    d. Select the fittest bee from each site.
    e. Assign remaining bees to search randomly and evaluate their fitness.
4. End While

3.2. Algorytm dla przedstawionego problemu

Poniżej przedstawiony został sposób działania algorytmu w przypadku naszego rozwiązania.
Wszystkie dane przechowywane są w macierzach zgodnych z definicjami z opisu zagadnienia.

3.2.1. Pseudokod algorytmu

W przypadku naszego problemu algorytm działa wg poniższego pseudokodu.
Wyróżnione nazwy są parametrami, które dokładnie opisane są w kolejnych rozdziałach.

1. Wygeneruj losową populację rozmiaru *population_size*
2. Dopóki warunek stopu nie jest spełniony
    2.1. Posortuj populację rosnąco względem kosztu
    2.2. Dla pierwszych *elite_sites* rozwiązań z populacji
        2.2.1. Wygeneruj *elite_site_size* rozwiązań z otoczenia tego rozwiązania
        2.2.2. Wybierz najlepsze z wygenerowanych rozwiązań i rozważanego rozwiązania
    2.3. Dla kolejnych *normal_sites* rozwiązań z populacji
        2.3.1. Wygeneruj *normal_site_size* rozwiązań z otoczenia tego rozwiązania
        2.3.2. Wybierz najlepsze z wygenerowanych rozwiązań i rozważanego rozwiązania
    2.4. Dla pozostałych rozwiązań z populacji
        2.4.1. Wygeneruj nowe losowe rozwiązanie
    2.5. Zastąp populację rozwiązaniami z punktów 2.2.2, 2.3.2 oraz 2.4.1
3. Zwróć pierwsze rozwiązanie z populacji

3.2.2. Generacja losowego rozwiązania

Generacja losowego rozwiązania przebiega zgodnie z poniższym pseudokodem, który gwarantuje że wygenerowan rozwiązanie będzie spełniać ograniczenia.

1. Każdej ciężarówce przypisz losowe przejście graniczne
2. Wygeneruj losowy przydział towarów do ciężarówek, korzystając z rozkładu wielomianowego tak aby drugie ograniczenie (suma towarów danych typów) było spełnione
3. Dopóki któraś ciężarówka ma więcej towarów niż jest w stanie przewieźć
    3.1. Znajdź najbardziej załadowaną ciężarówkę
    3.2. Znajdź najmniej załadowaną ciężarówkę
    3.3. Wybierz typ towaru którego jest najwięcej w pierwszej ciężarówce
    3.4. Przenieś tyle towaru wybranego typu ile się da z pierwszej do drugiej ciężarówki
4. Zwróć wygenerowane rozwiązanie

Algorytm ten korzysta zarówno z etapu losowego (1, 2) jak i heurystycznego (3).

3.2.3. Warunek stopu

Nasz algorytm pozwala na przekazanie dowolnego warunku stopu, natomiast zaimplementowane zostały dwa:

- Stop po określonej liczbie iteracji
- Stop gdy różnica kosztu pomiędzy ostatnimi dwoma iteracjami jest poniżej określonej wartośc

3.2.4. Generacja sąsiedniego rozwiązania

Generacja sąsiedniego rozwiązania przebiega zgodnie z poniższym pseudokodem, który gwarantuje że wygenerowane rozwiązanie będzie spełniać ograniczenia.

1. Utwórz kopię oryginalnego rozwiązania
2. Wykonaj *goods_mutations* razy
    2.1. Wybierz losowo typ towaru
    2.2. Wybierz losowo ciężarówkę, która posiada wolne miejsce
    2.3. Wybierz losowo ciężarówkę, która posiada wybrany towar
    2.4. Wybierz losowo ilość towaru, która jest mniejsza niż wolne miejsce w pierwszej ciężarówce oraz dostępny towar w drugiej ciężarówce
    2.5. Przenieś wybraną ilość towaru z drugiej ciężarówki do pierwszej
3. Wykonaj *trucks_mutations* razy
    3.1. Zmień przejście graniczne przypisane do losowej ciężarówki na inne losowe przejście graniczne
4. Zwróć rozwiązanie

3.2.5. Parametry algorytmu

Nasz algorytm posiada 7 parametrów: *population_size, goods_mutations, trucks_mutations, elite_sites, normal_sites, elite_site_size, normal_site_size.*
Wszystkie one przyjmują wartości całkowite, a ich dobór leży w gestii użytkownika. Ich dokładny opis oraz znaczenie jest opisany w kolejnych rozdziałach.

4. Aplikacja

Aplikacja zrealizowana została w języku Python i dostępna jest w formie biblioteki do programistycznego użytku. W tym rozdziale opiszemy jak z niej korzystać.

4.1. Wymagania

Aplikacja do działania wymaga Pythona w wersji 3.5+ oraz biblioteki Numpy, którą można zainstalować przy pomocy komendy:

```
> pip install numpy
```

Jako że nasza biblioteka nie jest dostępna w PyPi, należy ją ręcznie ściągnąć i umieścić w takim miejscu aby folder *goods_duty_optimier* znajdował się w miejscu z którego odpalany jest interpreter Pythona.

4.2. Definiowanie problemu do rozwiązania

Aby zdefiniować problem do optymalizacji musimy utworzyć obiekt typu *Settings*, który przechowuje wszystkie zmienne definiujące problem. Parametry przekazywane do konstruktora są następujące:

Nazwa parametru	Typ	Oznaczenie	Opis
crossings_number	int	n	Ilość przejść granicznych
goods_types_number	int	m	Ilość typów towarów
trucks_number	int	p	Ilość ciężarówek
truck_capacity	int	v	Pojemność każdej z ciężarówek
fuel_cost	float	f	Koszt paliwa
duties	np.array float (n, m)	c_{ij}	Cło za j-ty towar na i-tym przejściu
distances	np.array float (n,)	d_i	Długość trasy przez i-te przejście
goods_amounts	np.array int (m,)	t_j	Ilość j-tego towaru do przewiezienia

Przykładowe użycie:

```
# importy
import numpy as np
from goods_duty_optimizer import Settings

# macierz 3x4 opisująca cła
duties = np.array([
  [1.3, 2.3, 3.3, 7.2],
  [0.2, 1.1, 9.0, 4.1],
  [0.6, 3.4, 7.7, 2.2],
])

# wektor x3 opisujący dystanse
distances = np.array([1.2, 4.2, 6.0])

# wektor x4 opisujący ilości towarów
goods_amounts = np.array([10, 6, 14, 11])

# tworzenie ustawień modelu
settings = Settings(
  crossings_number=3,
  goods_types_number=4,
  trucks_number=5,
  truck_capacity=15,
  fuel_cost=5.3,
  duties=duties,
  distances=distances,
  goods_amounts=goods_amounts
)
```

4.3. Definiowanie parametrów algorytmu

Aby zdefiniować parametry używane podczas poszukiwania rozwiązania należy utworzyć obiekt typu *BeesSolver*. Parametry przekazywane do konstruktora są następujące:

Nazwa parametru	Typ	Opis
settings	Settings	Ustawienia problemu
population_size	int	Rozmiar populacji
goods_mutations	int	Ilość mutacji przydziału towarów przy wyznaczaniu sąsiada
trucks_mutations	int	Ilość mutacji przydziału ciężarówek przy wyznaczaniu sąsiada
elite_sites	int	Ilość rozwiązań wybieranych jako elitarne
normal_sites	int	Ilość rozwiązań wybieranych jako normalne
elite_site_size	int	Rozmiar przeszukiwanego sąsiedztwa dla rozwiązania elitarnego
normal_site_size	int	Rozmiar przeszukiwanego sąsiedztwa dla rozwiązania normalnego

Przykładowe użycie:

```
# importy
from goods_duty_optimizer import BeesSolver

# tworzenie solvera
solver = BeesSolver(
  settings=settings,
  population_size=10,
  goods_mutations=2,
  trucks_mutations=1,
  elite_sites=2,
  normal_sites=3,
  elite_site_size=7,
  normal_site_size=2
)
```

4.4. Generowanie rozwiązania

Mając zdefiniowany problem oraz parametry algorytmu możemy uruchomić poszukiwanie rozwiązania. Służy do tego metoda *BeesSolver.find_best_solution*, która przyjmuje jako jedyny argument warunek stopu. Dostępne są dwa warunki stopu:

stop_delta(d) - zatrzymuje optymalizację gdy poprawa kosztu między iteracjami jest mniejsza niż *d*
stop_iterations(n) - zatrzymuje optymalizację po *n* iteracjach

W rezultacie otrzymamy instancję klasy *Solution*, która udostępnia dwie wartości:

- *trucks_allocation* - wektor przypisujący ciężarówki do przejść granicznych
- *goods_allocation* - macierz przypisująca towary do ciężarówek

Aby wyliczyć koszt danego rozwiązania możemy skorzystać z funkcji *calculate_cost*, która przyjmuje rozwiązanie oraz ustawienia problemu.

Przykładowe użycie:

```
# importy
from goods_duty_optimizer import stop_delta, stop_iterations, calculate_cost

# znajdowanie rozwiązania z warunkiem stopu 'delta'
solution_delta = solver.find_best_solution(stop_delta(0.001))

# znajdowanie rozwiązania z warunkiem stopu 'iterations'
solution_iters = solver.find_best_solution(stop_iterations(1000))

# wypisanie rozwiązania
print("Solution delta - trucks:\n", solution_delta.trucks_allocation)
print("Solution delta - goods:\n", solution_delta.goods_allocation)
print("Solution delta - cost:", calculate_cost(solution_delta, settings))

print("Solution iters - trucks:\n", solution_iters.trucks_allocation)
print("Solution iters - goods:\n", solution_iters.goods_allocation)
print("Solution iters - cost:", calculate_cost(solution_iters, settings))
```

5. Testy

Wykonano trzy typy testów: manuale testy poprawności, testy wpływu parametrów na zbieżność oraz testy rozwiązań dla wybranych problemów. Poniżej opisane zostały dwie ostatnie grupy.

5.1. Wpływ parametrów na zbieżność

Testy te miały na celu zbadanie wpływu parametrów algorytmu na jego zbieżność do optymalnego rozwiązania. Tabelki przedstawiają minimalny i maksymalny wynik dla każdej wartości parametru po pięciokrotnym wykonaniu 1000 przebiegów optymalizacji
Wykresy przedstawiają krzywe kosztów dla różnych wartości badanego parametru na przestrzeni 1000 iteracji.

5.1.1. Parametry bazowe

Testy wykonywane były poprzez modyfikowanie każdego parametru po kolei podczas gdy pozostałe ustawiane były na wartości podane poniżej:

Nazwa parametru	Wartość	Opis
population_size	10	Rozmiar populacji
goods_mutations	2	Ilość mutacji przydziału towarów przy wyznaczaniu sąsiada
trucks_mutations	1	Ilość mutacji przydziału ciężarówek przy wyznaczaniu sąsiada
elite_sites	2	Ilość rozwiązań wybieranych jako elitarne
normal_sites	3	Ilość rozwiązań wybieranych jako normalne
elite_site_size	7	Rozmiar przeszukiwanego sąsiedztwa dla rozwiązania elitarnego
normal_site_size	2	Rozmiar przeszukiwanego sąsiedztwa dla rozwiązania normalnego

5.1.2. Badany problem

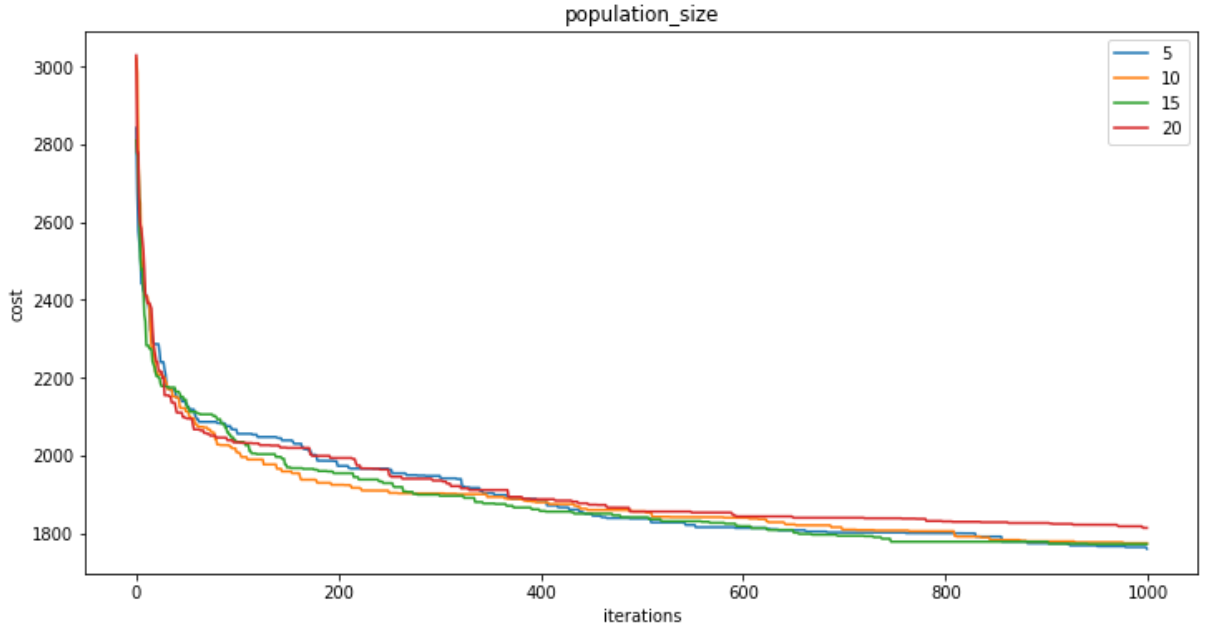
Wszystkie testy wykonywane zostały na jednym problemie o następujących parametrach:

Nazwa parametru	Wartość	Opis
crossings_number	10	Ilość przejść granicznych
goods_types_number	20	Ilość różnych typów produktów
trucks_number	15	Ilość ciężarówek
truck_capacity	29	Ładowność każdej z ciężarówek
fuel_cost	6.93	Koszt paliwa za jedną jednostkę odległości
duties	losowo	Cła za przewiezienie danego typu towaru przez dane przejście
distances	losowo	Odległości do poszczególnych przejść granicznych
goods_amounts	losowo	Ilości dóbr danych typów do przewiezienia

5.1.3. Testy parametru ‘population_size’

Parametr ten opisuje rozmiar symulowanej populacji.
W naszym przypadku może przyjąć wartości z zakresu [5, 6, 7, …]
Testy przeprowadzono dla wartości [5, 10, 15, 20]

Parametr	5	10	15	20
Min. koszt	1772.82	1765.73	1758.65	1760.21
Max. koszt	1797.44	1807.91	1803.20	1787.26



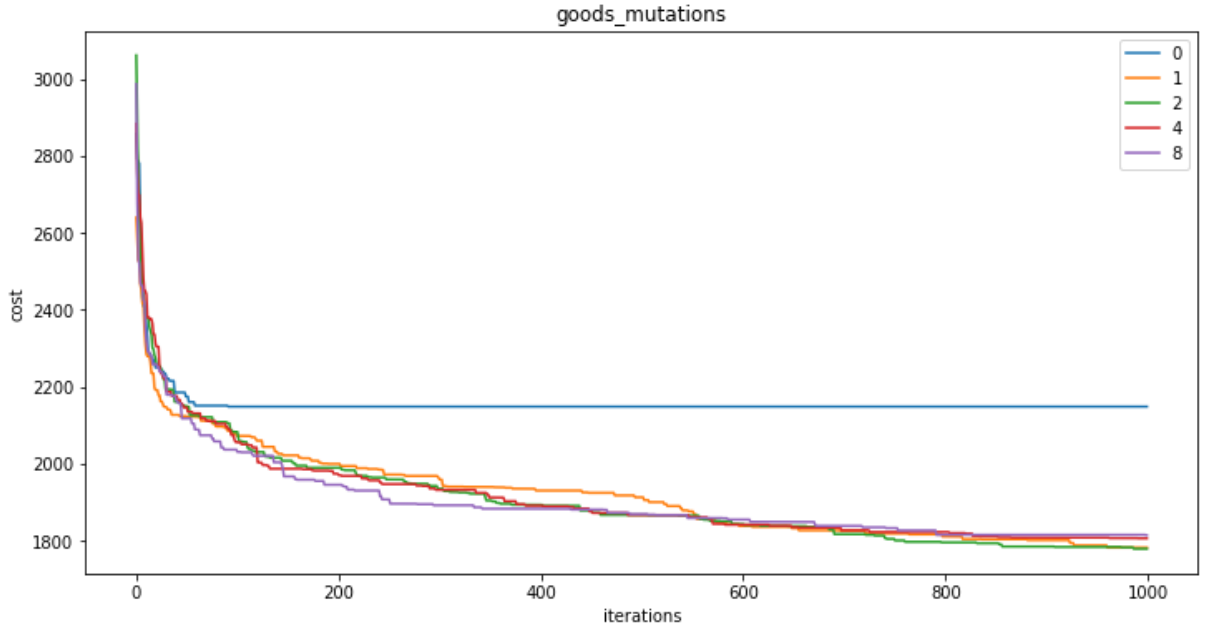
Jak widzimy rozmiar populacji nie ma większego wpływu na wyniki, co spowodowane jest tym, że przy naszych ustawieniach 5 najlepszych osobników z populacji jest traktowanych jako elitarne/normalne rozwiązania, a reszta jest losowana. A zatem zwiększenie rozmiaru populacji powyżej 5 zwiększa jedynie ilość losowych rozwiązań w każdej populacji, co nie ma większego wpływu gdyż nasz problem nie przejawia tendencji do posiadania wielu minimów lokalnych.

Jako że zwiększenie rozmiaru populacji zwiększa ilość obliczeń, możemy przyjąć że wartość **10** jest odpowiednia, ponieważ pozwala na losowe znalezienie lepszych rozwiązań, a przy tym nie wydłuża znacząco czasu obliczeń.

5.1.4. Testy parametru ‘goods_mutations’

Parametr ten opisuje ilość mutacji w przydziale towarów przy wyznaczaniu sąsiedniego rozwiązania.
Może przyjąć wartości z zakresu [0, 1, 2, ...]
Testy przeprowadzono dla wartości [0, 1, 2, 4, 8]

Parametr	0	1	2	4	8
Min. koszt	2139.02	1776.17	1742.65	1775.29	1811.20
Max. koszt	2176.79	1807.31	1785.23	1795.65	1860.32



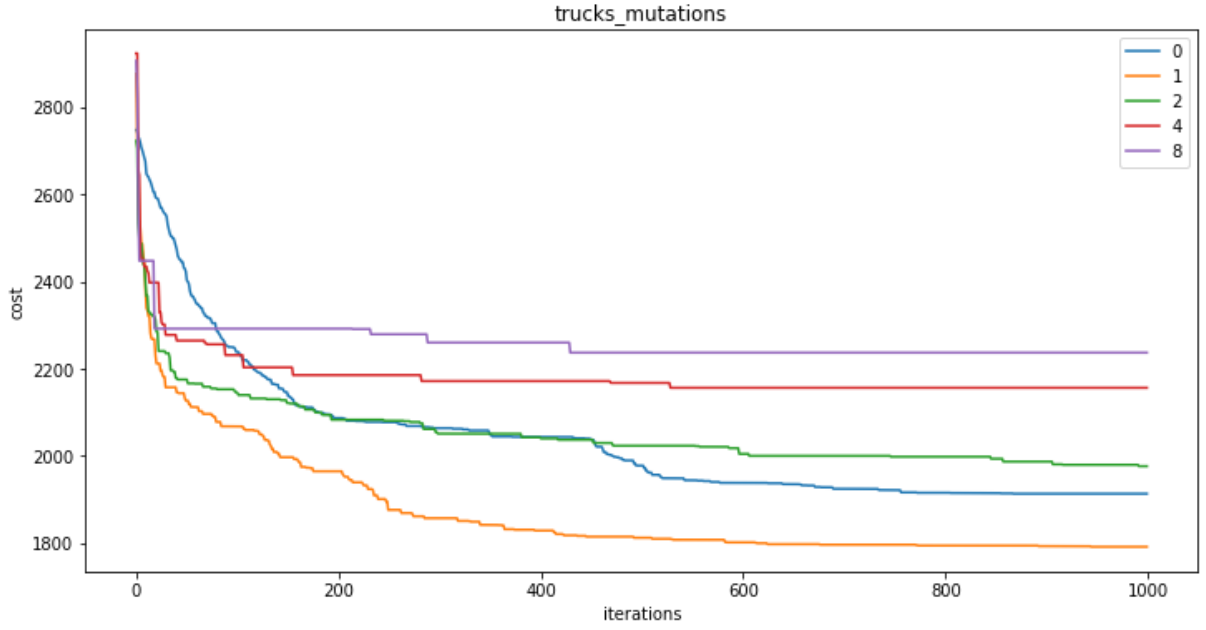
Pierwszym interesującym spostrzeżeniem jest to, że w przypadku braku mutacji produktów bardzo szybko dochodzi do stagnacji rozwiązania, jest to spowodowane tym że sama mutacja przydziału ciężarówek nie jest w stanie zbadać całej dziedziny rozwiązań.

Dla pozostałych wartości otrzymujemy dość porównywalne rezultaty, przy czym dla wartości **2** otrzymujemy delikatnie lepsze, więc możemy tą wartość przyjąć za najlepszą.

5.1.5. Testy parametru ‘trucks_mutations’

Parametr ten opisuje ilość mutacji w przydziale ciężarówek przy wyznaczaniu sąsiedniego rozwiązania.
Może przyjąć wartości z zakresu [0, 1, 2, …]
Testy przeprowadzono dla wartości [0, 1, 2, 4, 8]

Parametr	0	1	2	4	8
Min. koszt	1736.17	1759.48	1948.46	2111.78	2196.08
Max. koszt	1903.81	1808.99	1977.14	2136.52	2234.58



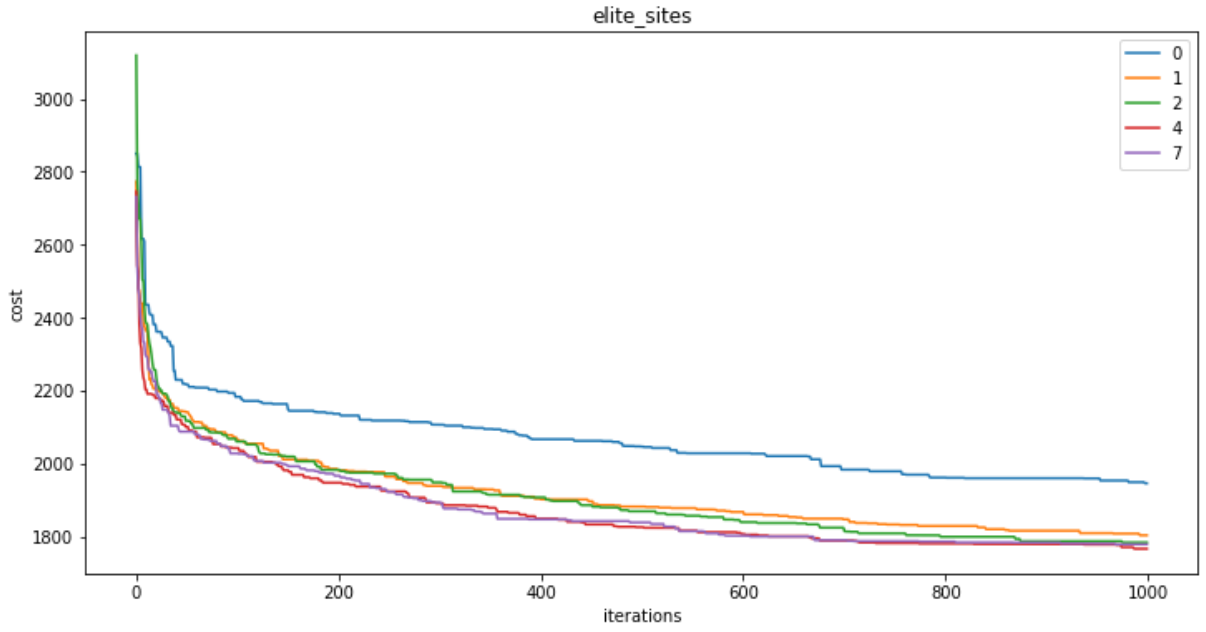
Generalne spostrzeżenie jest takie, że im więcej mutacji przydziału ciężarówek tym gorzej. Prawdopodobnie wynika to z tego, że ciągłe zmiany w tym zakresie utrudniają zoptymalizowanie przydziału towarów do poszczególnych ciężarówek, ponieważ cło za dany towar w danej ciężarówce jest cały czas zmieniany.

Natomiast brak mutacji też nie jest pożądany ponieważ nie pozwala na zbadanie większości możliwych rozwiązań, zatem możemy bezpiecznie założyć że wartość **1** jest najbardziej optymalna.

5.1.6. Testy parametru ‘elite_sites’

Parametr ten opisuje ile rozwiązań jest traktowanych jako elitarne.
W naszym przypadku może przyjąć wartości z zakresu [0, 1, …, 7]
Testy przeprowadzono dla wartości [0, 1, 2, 4, 7]

Parametr	0	1	2	4	7
Min. koszt	1874.55	1760.06	1757.32	1763.95	1764.91
Max. koszt	1939.39	1801.58	1783.71	1790.03	1772.07

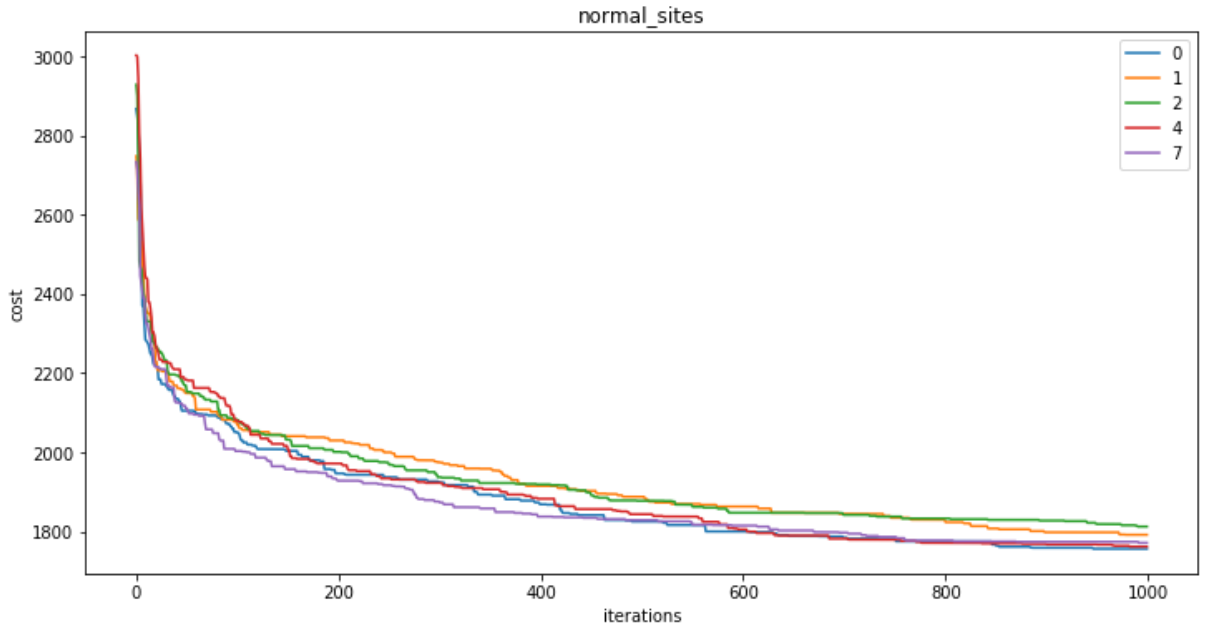


Jak widać im więcej elitarnych elitarnych miejsc tym algorytm zbiega szybciej do optymalnego rozwiązania. Natomiast różnica pomiędzy wartością 1 a 7 nie jest zbyt znacząca, a zwiększa ilość obliczeń w każdej iteracji, dlatego też możemy uznać że wartości z okolic **1**, **2** są najbardziej optymalne.

5.1.7. Testy parametru ‘normal_sites’

Parametr ten opisuje ile rozwiązań jest traktowanych jako normalne.
W naszym przypadku może przyjąć wartości z zakresu [0, 1, …, 8]
Testy przeprowadzono dla wartości [0, 1, 2, 4, 7]

Parametr	0	1	2	4	7
Min. koszt	1765.93	1740.23	1768.95	1743.36	1768.50
Max. koszt	1795.69	1776.45	1781.26	1778.63	1790.86



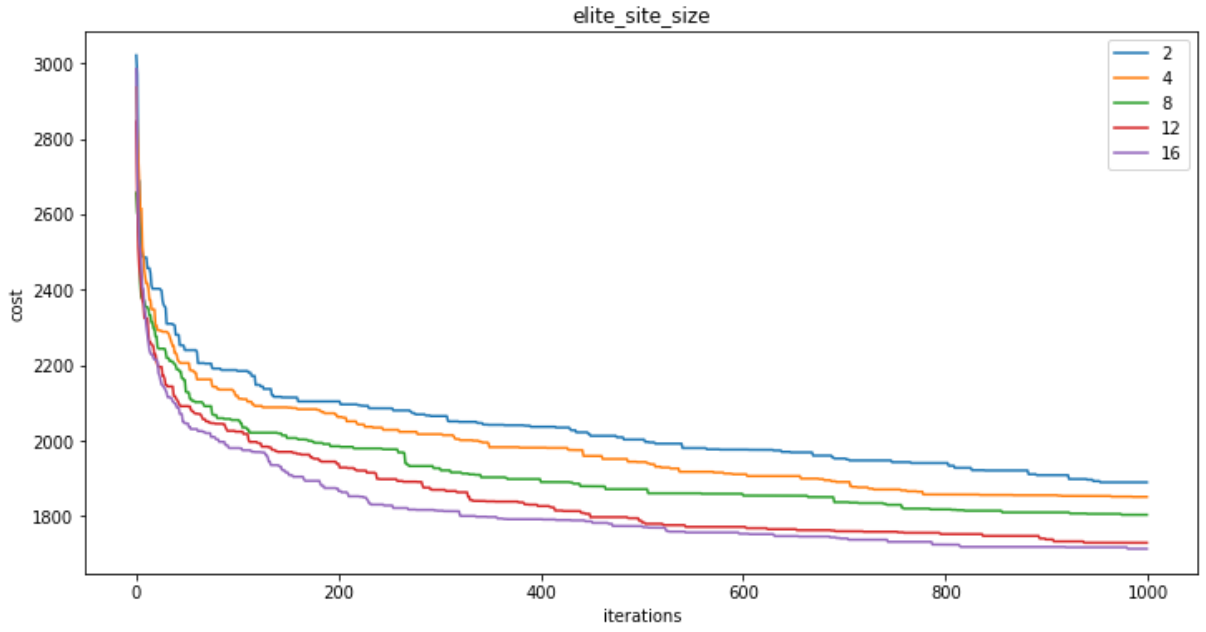
Jak widać parametr ten nie ma większego wpływu na zbieżność algorytmy, prawdopodobnie dla tego że to rozwiązania elitarne odgrywają główną rolę w redukcji kosztu.

Prawdopodobnie parametr ten miał by większe znaczenie gdyby różnica pomiędzy rozmiarem miejsc elitarnych i normalnych była większa, ale w naszym przypadku tak nie jest przez co parametr możemy uznać za nie znaczący i ustawić go na wartości z zakresu **0-4** żeby zmniejszyć ilość obliczeń.

5.1.8. Testy parametru ‘elite_site_size’

Parametr ten opisuje ilość sąsiednich rozwiązań generowanych dla każdego elitarnego rozwiązania.
Może przyjąć wartości z zakresu [0, 1, 2, ...]
Testy przeprowadzono dla wartości [2, 4, 8, 12, 16]

Parametr	2	4	8	12	16
Min. koszt	1877.43	1810.03	1758.32	1720.48	1712.03
Max. koszt	1913.44	1859.91	1784.05	1752.87	1749.95



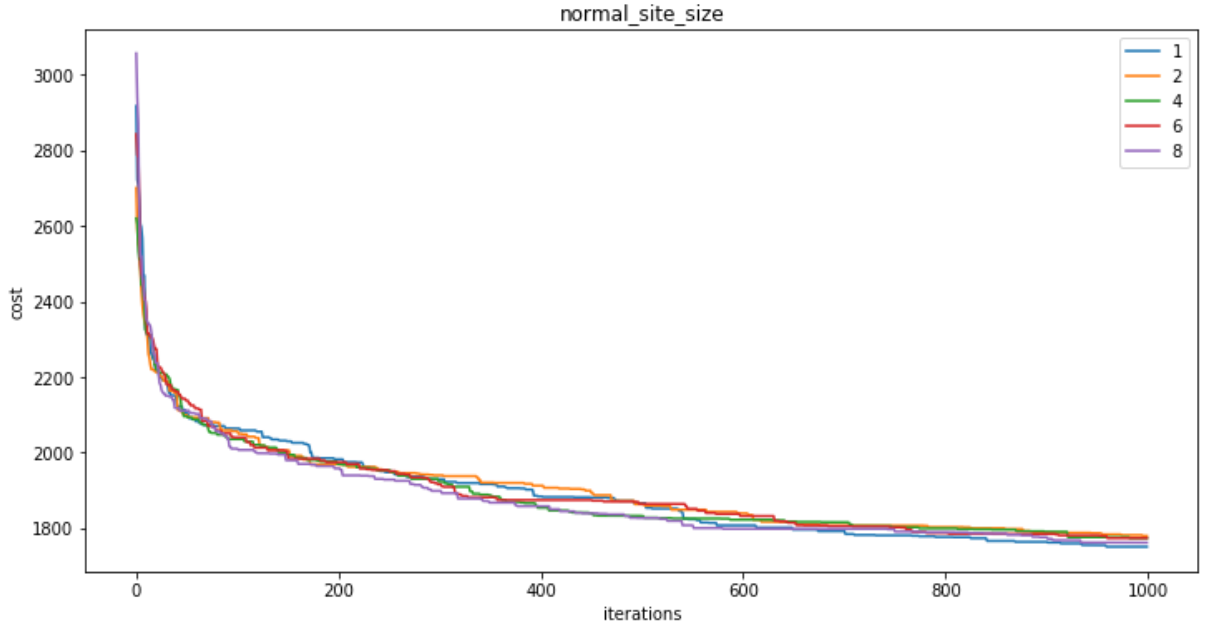
W przypadku tego parametru widzimy bezpośrednią zależność między jego wartością, a tempem zbiegania do rozwiązania optymalnego. Tak naprawdę ma on wpływ tylko na to czy wykonujemy mniej bardziej czasochłonnych iteracji, czy więcej mniej czasochłonnych.

A zatem możemy uznać że nie ma on wpływu na interesującą nas właściwość algorytmu i ustawiać go na względnie dowolną wartość.

5.1.9. Testy parametru ‘normal_site_size’

Parametr ten opisuje ilość sąsiednich rozwiązań generowanych dla każdego normalnego rozwiązania.
Może przyjąć wartości z zakresu [0, 1, 2, ...]
Testy przeprowadzono dla wartości [1, 2, 4, 6, 8]

Parametr	1	2	4	6	8
Min. koszt	1735.69	1764.41	1753.32	1756.33	1738.49
Max. koszt	1795.73	1797.85	1779.63	1801.92	1768.57



Podobnie jak w przypadku parametru *normal_sites*, ten parametr nie ma znaczącego wpływu na zbieżność ponieważ to elitarne miejsca odgrywają znaczącą rolę w redukcji kosztu. A zatem możemy go ustawiać na niskie wartości tak aby zredukować liczbę obliczeń.

5.2. Rozwiązania dla wybranych przykładów

W czasie tej fazy testów sprawdziliśmy algorytm przy pomocy pięciu różnych problemów, z których każdy testowany był pięciokrotnie.
Parametry algorytmu zostały dobrane ręcznie zgodnie z informacjami uzyskanymi w poprzedniej fazie testów.

5.2.1. Testowane problemy

Poniżej znajduje się podsumowanie parametrów poszczególnych testowanych problemów.

	mały problem	**średni problem**	**duży problem**	**problem ceł**	**problem dystansów**
crossings_number	3	10	20	3	3
goods_types_number	5	20	40	3	1
trucks_number	5	15	25	3	3
truck_capacity	15	29	25	1000	10
fuel_cost	0.66	6.93	8.93	0.00	1.00
duties	losowo	losowo	losowo	*	[0,0,0]
distances	losowo	losowo	losowo	[1,1,1]	[10,1,10]
goods_amounts	losowo	losowo	losowo	[100,100,100]	[25]

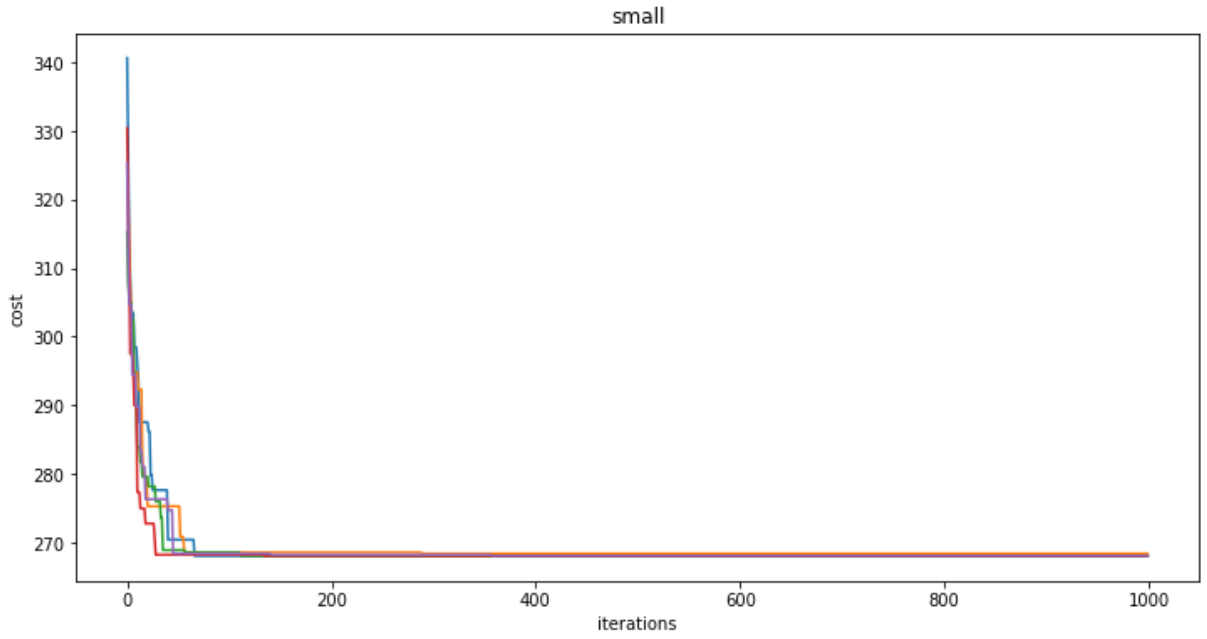
* macierz wypełniona wartościami 10.0, oprócz przekątnej na której znajdują się 1.0

5.2.1. Testy dla małego problemu

Problem ten miał za zadanie sprawdzić zachowanie algorytmu dla małej ilości losowych danych.

Wyniki przedstawiają się następująco:

Numer testu	1	2	3	4	5
Czas 1000 iteracji	3.70	3.63	3.60	3.62	3.63
Końcowy koszt	268.05	268.39	268.05	268.05	268.05



Jak widać dla tak małego problemu algorytm znajduje minimum już po ok 60 iteracjach.

Po ręcznym przeglądnięciu wyników możemy zauważyć że optymalnym rozwiązaniem jest wysłanie trzech ciężarówek do przejścia granicznego nr 2 i dwóch do przejścia nr 0:

[0, 2, 2, 2, 0]

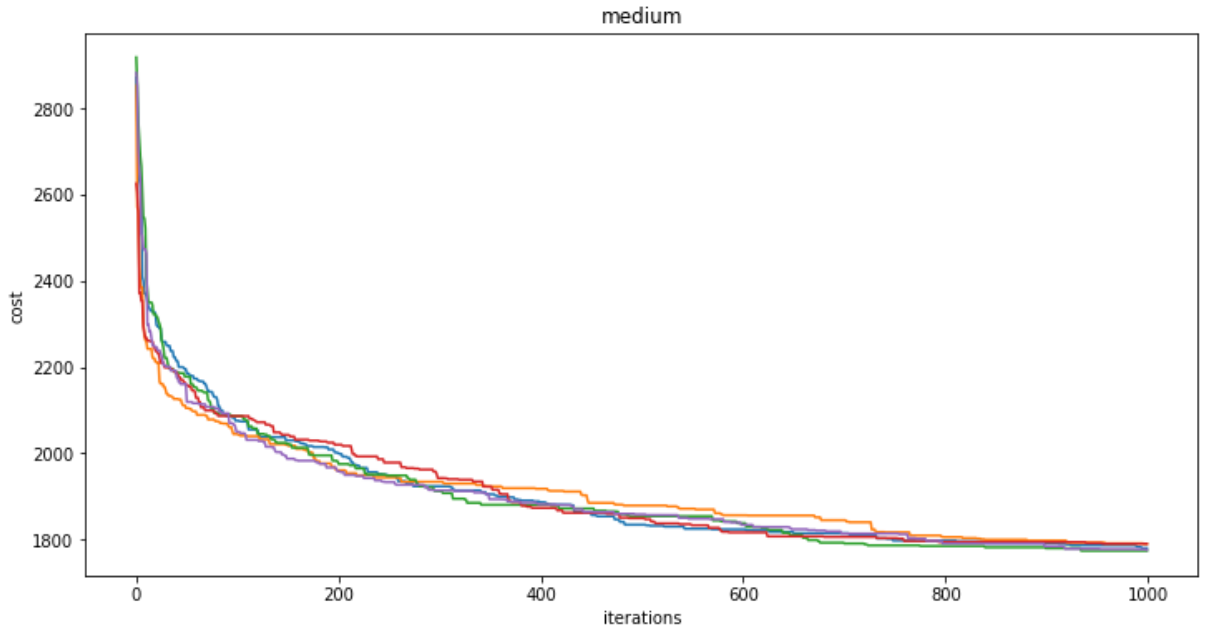
Dzieje się tak ponieważ przejście graniczne nr 2 jest zdecydowanie bliżej niż dwa pozostałe, natomiast ma znacznie wyższe cła dla produktów typu 1 i 2. A zatem algorytm generuje poprawne rozwiązanie ponieważ wysyła towary drogie na przejściu nr 2 przez przejście nr 1, a wszystkie pozostałem przez nr 2 tym samym redukując koszt paliwa.

5.2.2. Testy dla średniego problemu

Problem ten miał za zadanie sprawdzić zachowanie algorytmu dla średniej ilości losowych danych.

Wyniki przedstawiają się następująco:

Numer testu	1	2	3	4	5
Czas 1000 iteracji	4.98	4.64	4.83	4.63	4.67
Końcowy koszt	1780.25	1789.79	1774.06	1790.43	1775.88



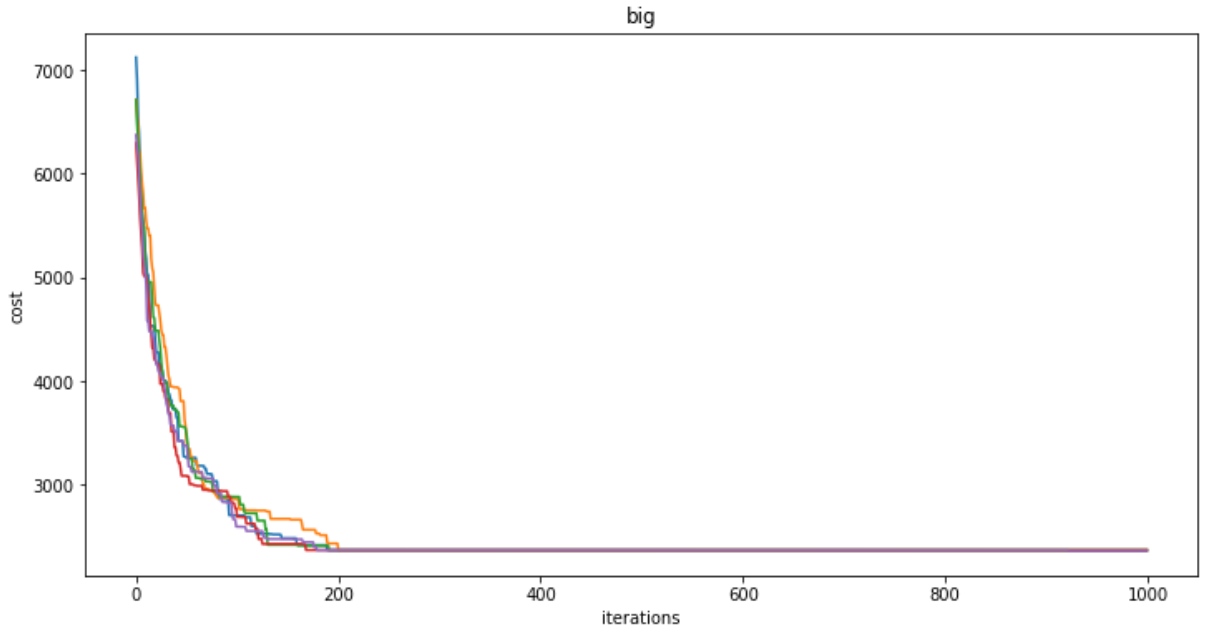
W przypadku tego problemu ciężko jest dokonać ręcznej analizy, ponieważ mamy do czynienia ze zbyt dużą ilością danych. Jednak przeglądając wyniki możemy zauważy,ć że algorytm zawsze wysyła ciężarówki tylko przez przejścia nr 0, 4, 6 i 7, całkowicie pomijając pozostałe. Ma to sens ponieważ w tym problemie mamy stosunkowo wysoki koszt paliwa, a przejścia 0, 4, 6 i 7 są przejściami o najmniejszych dystansach.

5.2.3. Testy dla dużego problemu

Problem ten miał za zadanie sprawdzić zachowanie algorytmu dla dużej ilości losowych danych.

Wyniki przedstawiają się następująco:

Numer testu	1	2	3	4	5
Czas 1000 iteracji	5.97	5.89	5.95	5.91	6.04
Końcowy koszt	2372.59	2372.59	2372.59	2372.59	2366.15



Podobnie jak w średnim problemie mamy do czynienia z problemem w którym koszt paliwa ma decydujący wpływ na całkowity koszt, tyle że tym razem jest to jeszcze bardziej widoczne ponieważ przejście graniczne nr 4 ma nieporównywalnie mniejszy dystans.
Skutkuje to tym, że optymalnym rozwiązaniem jest wysłanie wszystkich ciężarówek przez to przejście, a zatem przydział towarów pomiędzy ciężarówki nie ma żadnego znaczenia.

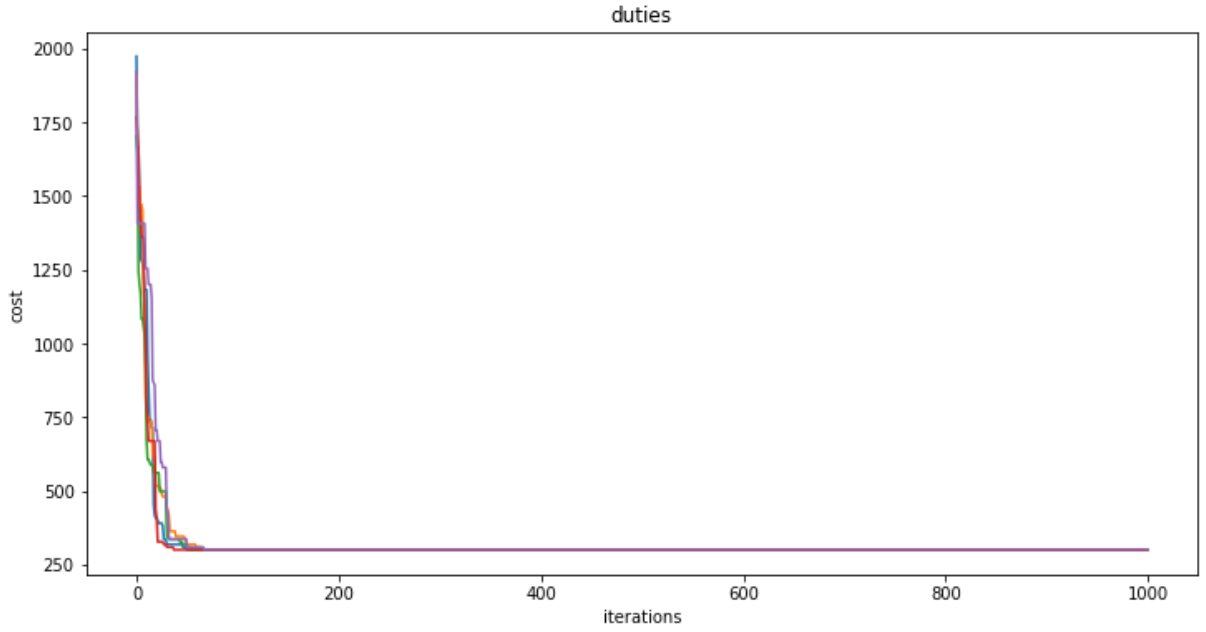
5.2.4. Testy dla problemu o dużej różnicy ceł

Problem ten miał za zadanie sprawdzić zachowanie algorytmu dla problemu, w którym mamy do czynienia z dużą różnicą ceł za poszczególne towary pomiędzy przejściami granicznymi.

Wartości w problemie zostały dobrane ręcznie, tak aby najoptymalniejszym rozwiązaniem było wysłanie trzech ciężarówek do trzech różnych przejść, gdzie każda z ciężarówek wiezie dokładnie jeden typ produktów.

Wyniki przedstawiają się następująco:

Numer testu	1	2	3	4	5
Czas 1000 iteracji	3.55	3.54	3.56	3.58	3.51
Końcowy koszt	300.00	300.00	300.00	300.00	300.00



Uzyskanym wynikiem jest dokładnie to czego oczekiwaliśmy, przydział ciężarówek jest następujący:

[0, 1, 2]

Przydział towarów:

[100, 0, 0]
[0, 100, 0]
[0, 0, 100]

Jest to najoptymalniejsze możliwe rozwiązanie, a zatem algorytm działa poprawnie.

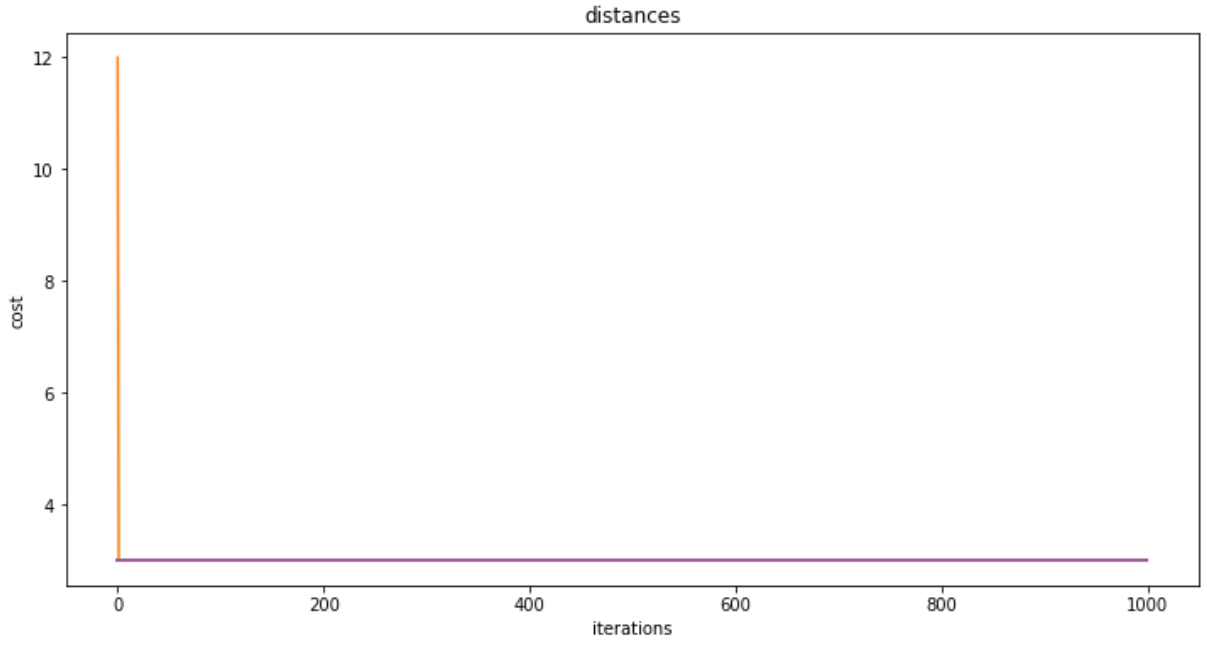
5.2.5. Testy dla problemu o dużej różnicy odległości

Problem ten miał za zadanie sprawdzić zachowanie algorytmu dla problemu, w którym mamy do czynienia z dużą różnicą odległości do poszczególnych przejść granicznych.

Wartości w problemie zostały ustawione ręcznie, tak aby najoptymalniejszym rozwiązaniem było wysłanie wszystkich ciężarówek przez jedno przejście.

Wyniki przedstawiają się następująco:

Numer testu	1	2	3	4	5
Czas 1000 iteracji	3.46	3.44	3.40	3.42	3.52
Końcowy koszt	3.00	3.00	3.00	3.00	3.00



Zgodnie z oczekiwaniami algorytm zdecydował się wysłać wszystkie ciężarówki przez przejście nr 1:

[1, 1, 1]

W takim przypadku przydział towarów do ciężarówek nie ma znaczenia, przez co najoptymalniejsze rozwiązanie znajdowane jest w zaledwie kilku iteracjach.

6. Podsumowanie

Jak widać utworzony przez nas algorytm osiąga bardzo zadowalające wyniki, robiąc to w dodatku sensownym czasie. Jego podstawową zaletą jest również to, że jest łatwy w implementacji i zrozumieniu, szczególnie gdy porównamy go do deterministycznych alternatyw głównie opierających się na układach równań i nierówności liniowych.

Podczas pracy nad projektem napotkaliśmy dwa istotne problemy. Pierwszym okazał się sposób reprezentacji danych. Początkowo w macierzy przydziału towarów używaliśmy liczb zmiennoprzecinkowych, co po kilkunastu mutacjach tej macierzy powodowało, że nie spełniała ona już ograniczenia na sumę kolumn, co spowodowane było oczywiście niedokładnością operacji zmiennoprzecinkowych. Aby rozwiązać ten problem zmieniliśmy reprezentację na całkowitoliczbową, nie wpłynęło to na sam problem, a uprościło implementację.

Drugim napotkanym problemem było sprawdzenie poprawności wyników. Ponieważ mamy do czynienia z trudnym problemem optymalizacyjnym dla którego nie znamy deterministycznego algorytmu mogliśmy dokonywać jedynie ręcznej weryfikacji poprawności dla małych problemów testowych.

Ogólnie rzecz biorąc projekt można uznać za udany jako że w sensownym czasie potrafił znaleźć poprawne rozwiązanie dla zadanego problemu.

7. Literatura

[1] Pham, D. & Ghanbarzadeh, Afshin & Koç, Ebubekir & Otri, Sameh & Rahim, Sahra & Zaidi, Mb. (2005). The Bees Algorithm Technical Note. Manufacturing Engineering Centre, Cardiff University, UK. 1–57.