Badania Operacyjne

Projekt

Szymon Budziak
Alicja Hurbol
Jakub Szymczak

1. Opis projektu

1.1. Cel oraz założenia projektu

Zadaniem jest zminimalizowanie kosztu dostarczenia na Marsa wszystkich modułów bazy kosmicznej. Pod uwagę brane jest ilość rakiet, ich pojemność, paliwo przez nie spalane oraz dodatkowe paliwo spalane przy przewożeniu danego modułu przez konkretny typ rakiety.
W celu znalezienia optymalnego rozwiązania został wykorzystany algorytm oparty na algorytmie pszczelim.

1.2. Opis problemu

Za pomocą konkretnej ilości rakiet, o konkretnej pojemności, należy przewieźć zadaną ilość modułów na Marsa. Typy rakiet mają różne bazowe spalanie paliwa oraz dodatkowe zależne od przewożonych modułów.
Celem jest wybranie kombinacji rakiet, która spali jak najmniej paliwa przewożąc wszystkie rakiety.

1.3. Możliwe rozszerzenia problemu

W projekcie umieszczony jest ograniczony problem z racji na ograniczenia czasowe. Można by go rozszerzyć np o:

a. objętość konkretnych modułów
b. zakazane kombinacje przewożonych modułów w danej rakiecie

2. Model matematyczny

Matematyczny sposób sformułowania naszego problemu.

2.1. Dane

- ★ $n \in \mathbb{N}$ - ilość typów rakiet
- ★ $m \in \mathbb{N}$ - ilość typów modułów
- ★ $p \in \mathbb{N}$ - ogólna ilość rakiet
- ★ $v \in \mathbb{R}$ - ładowność jednej rakiety
- ★ $di \in \mathbb{R}$ - ilość paliwa zużywana przez daną rakietę
- ★ $cij \in \mathbb{R}$ - dodatkowa ilość paliwa zależna od modułu i rakiety
- ★ $tj \in \mathbb{R}$ - ilość danego typu modułu, który trzeba przewieźć

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

2.2. Szukane

- ★ $ak \in \mathbb{N}$ - jakiego typu jest k-ta rakieta
- ★ $bkj \in \mathbb{R}$ - ilość j-tych modułów do przewiezienia w k-tej rakiecie

2.3. Ograniczenia

★ ograniczenie ładowności rakiety $\forall_k \sum_j b_{kj} \leq v$

★ bkj $\in \mathbb{R}$ - ilość j-tych modułów do przewiezienia w k-tej rakiecie $\forall_j \sum_k b_{kj} = t_j$

2.4. Minimalizowana funkcja

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

3. Opis algorytmu

3.1. Idea i algorytm pszczeli

Do rozwiązania naszego problemu zastosowaliśmy algorytm pszczeli.
W podstawowej wersji algorytm ten polega na wylosowaniu początkowej puli rozwiązań, 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 algorytmu pszczelego:

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

3.2. Algorytm dla przedstawionego problemu

3.2.1. Pseudokod algorytmu

1. Wygeneruj losowej populacji rozmiaru **population_size**
2. Dopóki warunek stopu nie jest spełniony
    a. Posortuj populacji rosnąco względem kosztu
    b. Dla pierwszych **elite_sites** rozwiązań wybierz najlepsze rozwiązanie z wygenerowanych **elite_site_size** rozwiązań z otoczenia tego rozwiązania (włącznei z nim)
    c. Dla kolejnych **normal_sites** rozwiązań wybierz najlepsze rozwiązanie z wygenerowanych **normal_site_size** rozwiązań z otoczenia tego rozwiązania (włącznei z nim)
    d. Dla pozostałych rozwiązań wygeneruj nowe losowe rozwiązania
    e. Zastąp populacje rozwiązaniami z punktów b, c, d
3. Zwróć pierwsze rozwiązanie z populacji

3.2.2. Generacja losowego rozwiązania

1. Dla każdej rakiety wybierz losowy typ
2. Wygeneruj losowy rozdział modułów dla rakiet (rozdział wielomianowy) - tak, aby drugie ograniczenie było spełnione
3. Dopóki dla któreś rakiety nie jest spełnione ograniczenie pierwsze
    a. Znajdź najmniej i najbardziej załadowaną rakiete
    b. Wybierz rodzaj modułu którego jest najwięcej w drugiej rakiecie
    c. Przenieś tyle ile się da tego towaru miedzy ciężarówkami
4. Zwróć rozwiązanie

3.2.3. Generacja sąsiedniego rozwiązania

1. Utwórz kopię rozwiązania
2. Wykonaj **modules_mutations** razy
    a. Wybierz losowy rodzaj modułu
    b. Wybierz losową rakietę, która ma wolne miejsce
    c. Wybierz losową rakietę, która ma wybrany towar
    d. Wybierz losową ilość modelu, która zmieścu się w rakiecie z wolnym miejscem
    e. Przenieś moduł
3. **rockets_type_mutations** razy zmień typ rakiety na losowo wybrany typ
4. Zwróć rozwiązanie

3.2.4. Warunek stopu

Nasz algorytm zatrzymuje się po określonej liczbie iteracji.

3.2.5. Parametry algorytmu

Algorytm posiada parametry (wszystkie są liczbami całkowitymi):

1. population_size
2. modules_mutations
3. rockets_type_mutations
4. elite_sites
5. normal_sites
6. elite_site_size
7. normal_site_size

4. Implementacja

4.1. Wymagania

Zaimplementowaliśmy algorytm w języku Python w wersji 3.8+, przy użyciu biblioteki Numpy, którą można zainstalować przy pomocy komendy:

```
pip install numpy
```

4.2. Definiowanie problemu do rozwiązania

Do zdefiniowania problemu potrzebujemy obiekt typu Settings, który posiada parametry:

Nazwa parametru	Oznaczenie	Typ	Opis
num_rocket_types	n	int	liczba typów rakiet
num_module_types	m	int	liczba typów modułów
num_rockets	p	int	ogólna liczba rakiet
rocket_capacity	v	int	ładowność jednej rakiety
fuel_costs	di	np.array float (n,)	ilość paliwa zużywana przez daną rakietę
additional_fuel_costs	cij	np.array float (n, m)	dodatkowa ilość paliwa zależna od modułu i rakiety
module_amounts	tj	np.array int (m,)	ilość danego typu modułu, który trzeba przewieźć

4.3. Definiowanie parametrów algorytmu

Do znalezienia rozwiązania potrzebujemy obiekt typu BeesSolver, który posiada parametry:

Nazwa parametru	Typ	Opis
settings	Settings	ustawienia problemu
population_size	int	rozmiar populacji
modules_mutations	int	liczba mutacji przydziału modułów przy wyznaczaniu sąsiada
rockets_type_mutations	int	liczba mutacji typów rakiet przy wyznaczaniu sąsiada
elite_sites	int	liczba rozwiązań wybieranych jako elitarne
normal_sites	int	liczba rozwiązań wybieranych jako normalne
elite_site_size	int	rozmiar przeszukiwanego sąsiedztwa dla rozwiązania elitarnego
normal_site_size	int	rozmiar przeszukiwanego sąsiedztwa dla rozwiązania normalnego

4.4. Generowanie rozwiązania

Mając zdefiniowany problem i parametry algorytmu można uruchomić poszukiwanie rozwiązania za pomocą metody **BeesSolver.find_best_solution,** podając ilość iteracji po jakich algorytm ma rzestać szukać.

W wyniku dostaniemy obiekt typu Solution, posiadający dwie wartości: **rocket_type_allocation** oraz **module_allocation** (tablice przypisanych typów rakiet oraz drugą z przypisaną liczbą danego typu rodzaju do rakiet).

5. Testy - wpływ parametrów na zbieżność

5.1. 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	12	rozmiar populacji
modules_mutations	5	liczba mutacji przydziału modułów przy wyznaczaniu sąsiada
rockets_type_mutations	1	liczba mutacji typów rakiet przy wyznaczaniu sąsiada
elite_sites	2	liczba rozwiązań wybieranych jako elitarne
normal_sites	3	liczba rozwiązań wybieranych jako normalne
elite_site_size	2	rozmiar przeszukiwanego sąsiedztwa dla rozwiązania elitarnego
normal_site_size	4	rozmiar przeszukiwanego sąsiedztwa dla rozwiązania normalnego

5.2. Badany problem

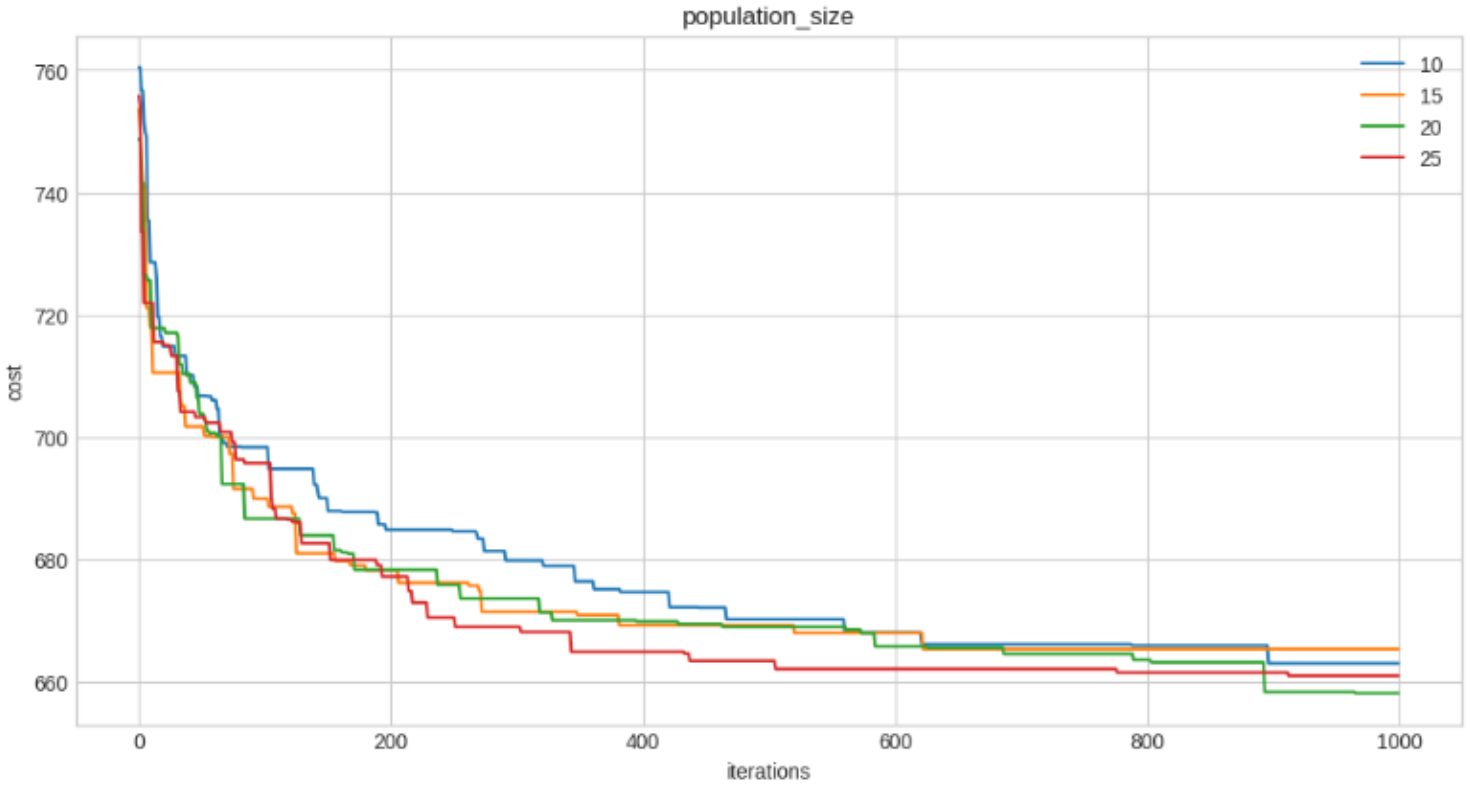
Wszystkie testy wykonywane zostały na jednym problemie o następujących parametrach:

Nazwa parametru	Wartość	Opis
num_rocket_types	7	liczba typów rakiet
num_module_types	8	liczba typów modułów
num_rockets	8	ogólna liczba rakiet
rocket_capacity	10	ładowność jednej rakiety
fuel_costs	[38.3, 37.5, 37.1, 36.9, 38.1, 37.7, 37.3]	ilość paliwa zużywana przez daną rakietę
additional_fuel_costs	[[8.76, 3.83, 5.97, 6.99, 8.93, 6.34, 8.16, 4.76], [6.18, 3.36, 5.5, 5.73, 8.55, 3.83, 8.33, 9.39], [6.08, 3.98, 5.55, 9.11, 5.49, 3.86, 7.91, 3.31], [7, 3, 7, 9, 4, 3, 10, 7], [9.35, 3.64, 7.63, 9.99, 3.9, 9.55, 7.36, 4.86], [9.96, 7.33, 3.31, 8.53, 8.07, 6.63, 7.42, 7.93], [5.22, 8.53, 6.99, 5.29, 4.1, 8.67, 8.14, 9.58]]	dodatkowa ilość paliwa zależna od modułu i rakiety
module_amounts	[6, 15, 9, 5, 1, 12, 20, 8]	ilość danego typu modułu, który trzeba przewieźć

5.3. Testy parametru population_size

Parametr ten opisuje rozmiar symulowanej populacji.
Testy przeprowadzono dla wartości [5, 10, 15, 20]

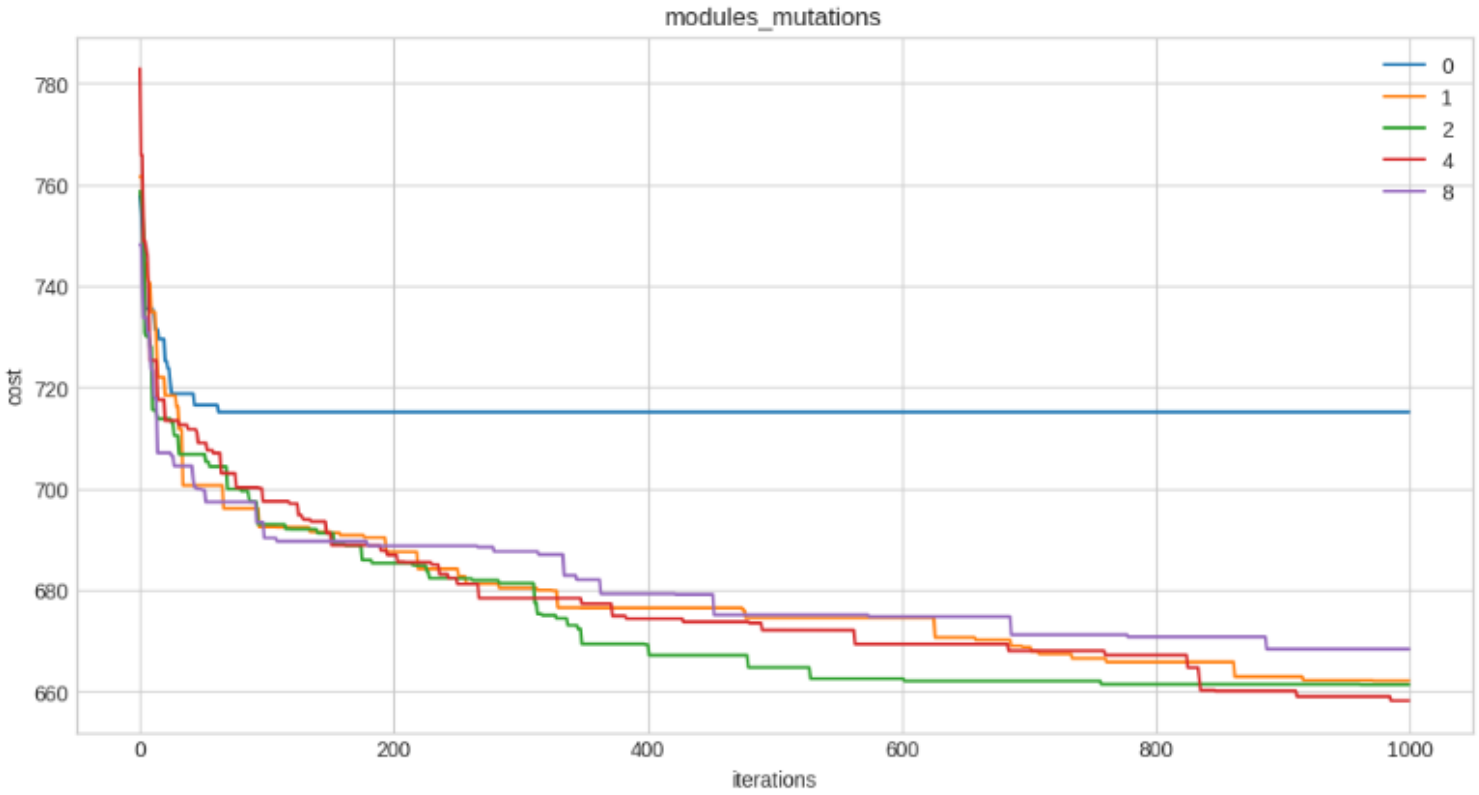
Wartość parametru	Minimalny koszt
5	663.0
10	665.33
15	658.11
20	660.97



5.4. Testy parametru modules_mutations

Parametr ten opisuje ilość mutacji przydziału modułów przy wyznaczaniu sąsiedniego rozwiązania.
Testy przeprowadzono dla wartości [0, 1, 2, 4, 8]

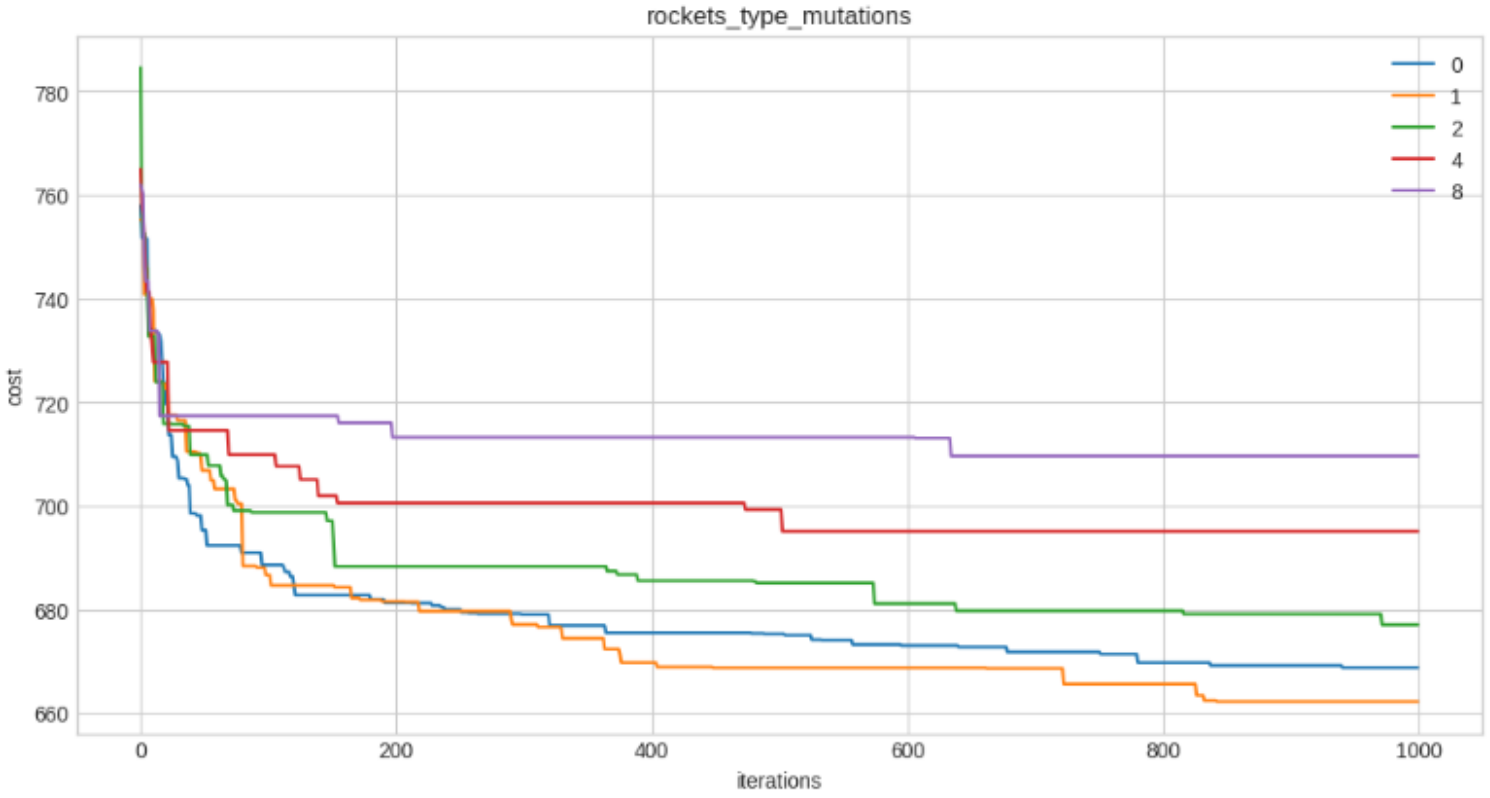
Wartość parametru	Minimalny koszt
0	715.12
1	662.06
2	661.32
4	658.13
8	668.3299999999999



5.5. Testy parametru rockets_type_mutations

Parametr ten opisuje ilość mutacji przydziału typów rakiet przy wyznaczaniu sąsiedniego rozwiązania.
Testy przeprowadzono dla wartości [0, 1, 2, 4, 8]

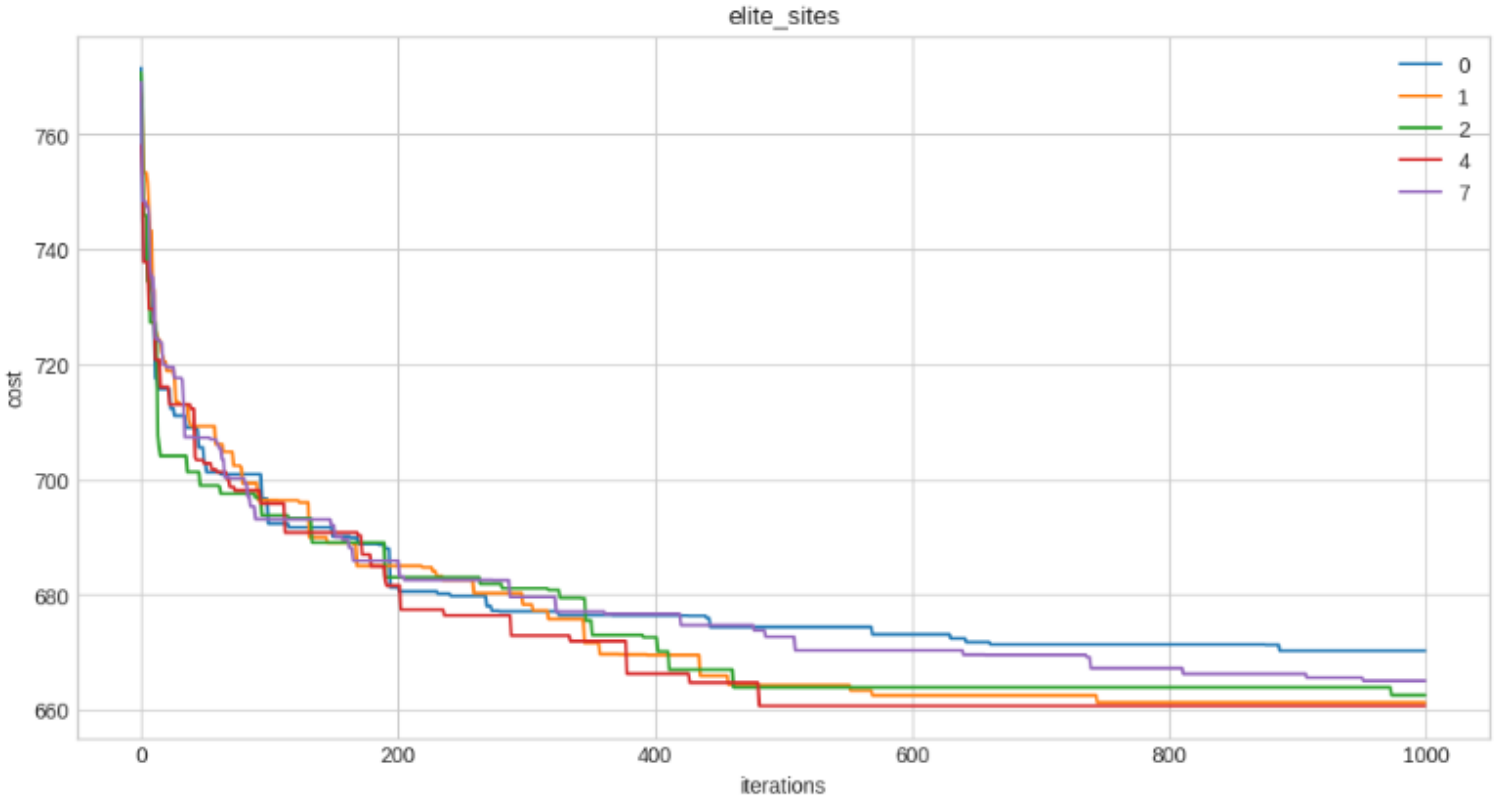
Wartość parametru	Minimalny koszt
0	668.75
1	662.21
2	677.03
4	695.0799999999999
8	709.5999999999999



5.6. Testy parametru elite_sites

Parametr ten opisujeile rozwiązań jest traktowanych jako elitarne.
Testy przeprowadzono dla wartości [0, 1, 2, 4, 7]

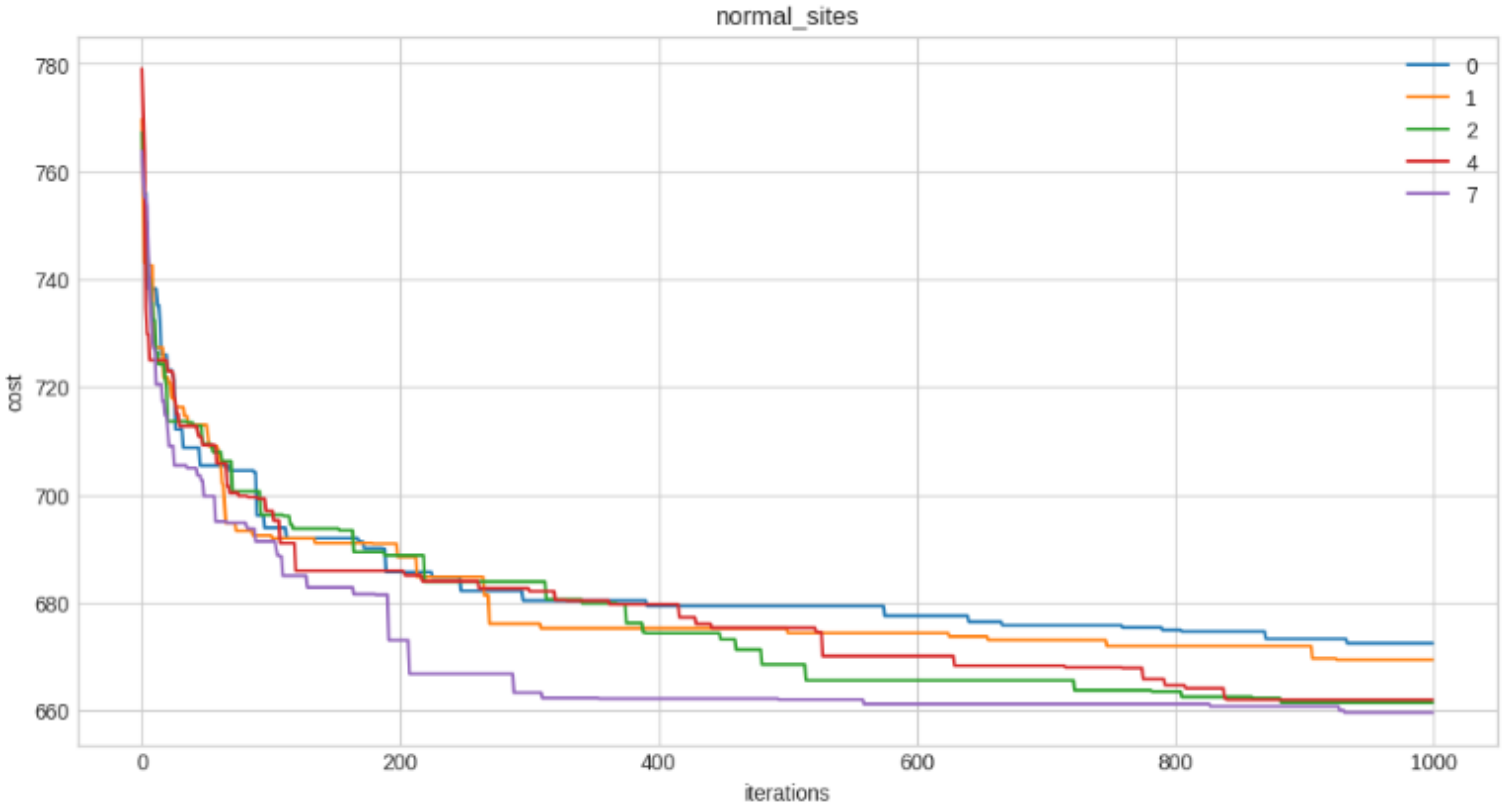
Wartość parametru	Minimalny koszt
0	670.28
1	661.29
2	662.5699999999999
4	660.7
7	665.0699999999999



5.7. Testy parametru normal_sites

Parametr ten opisuje ile rozwiązań jest traktowanych jako normalne.
Testy przeprowadzono dla wartości [0, 1, 2, 4, 7]

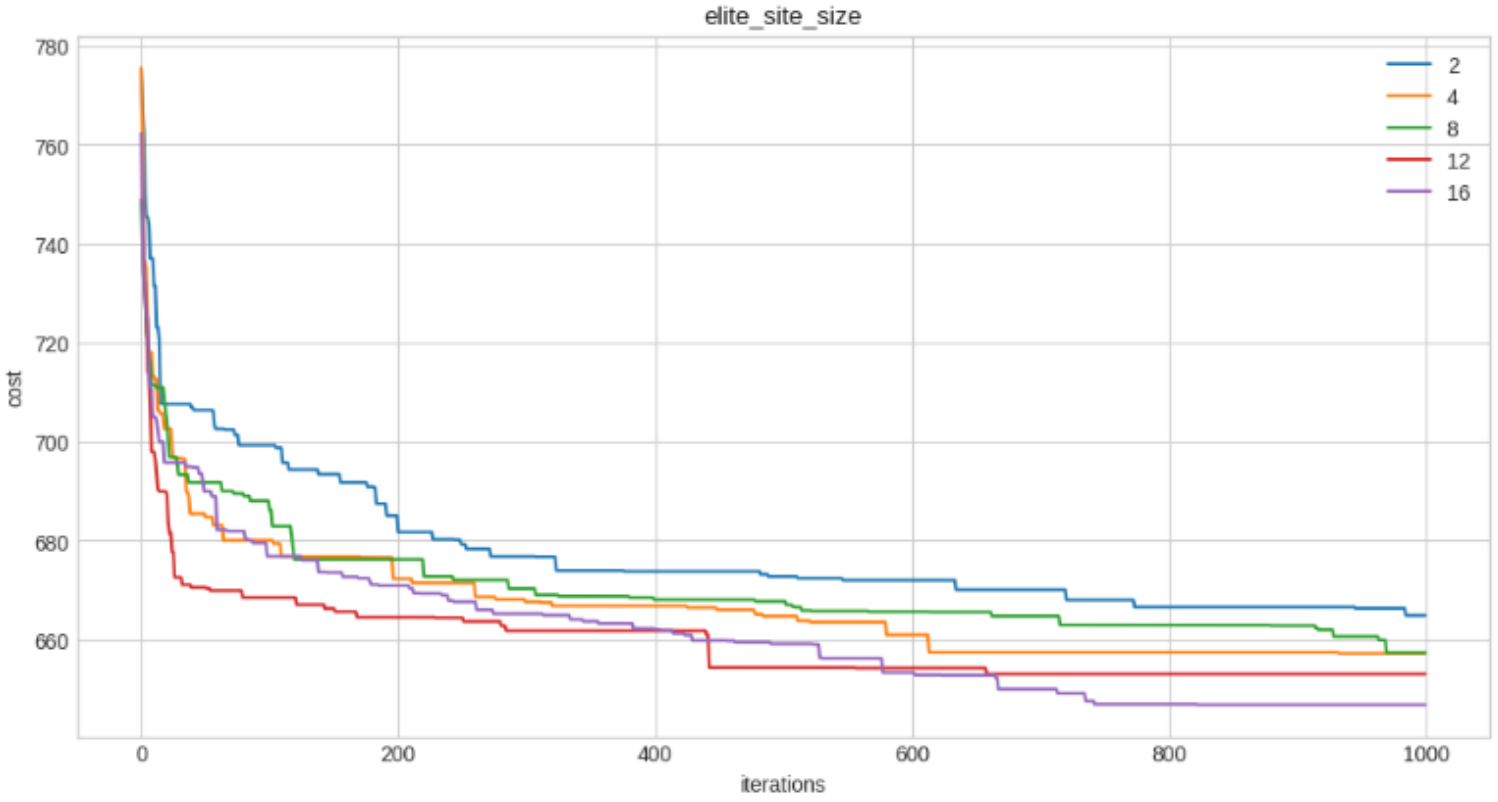
Wartość parametru	Minimalny koszt
0	672.44
1	669.35
2	661.4200000000001
4	661.95
7	659.5799999999999



5.8. Testy parametru elite_site_size

Parametr ten opisuje ilość sąsiednich rozwiązań generowanych dla każdego elitarnego rozwiązania
Testy przeprowadzono dla wartości [2, 4, 8, 12, 16]

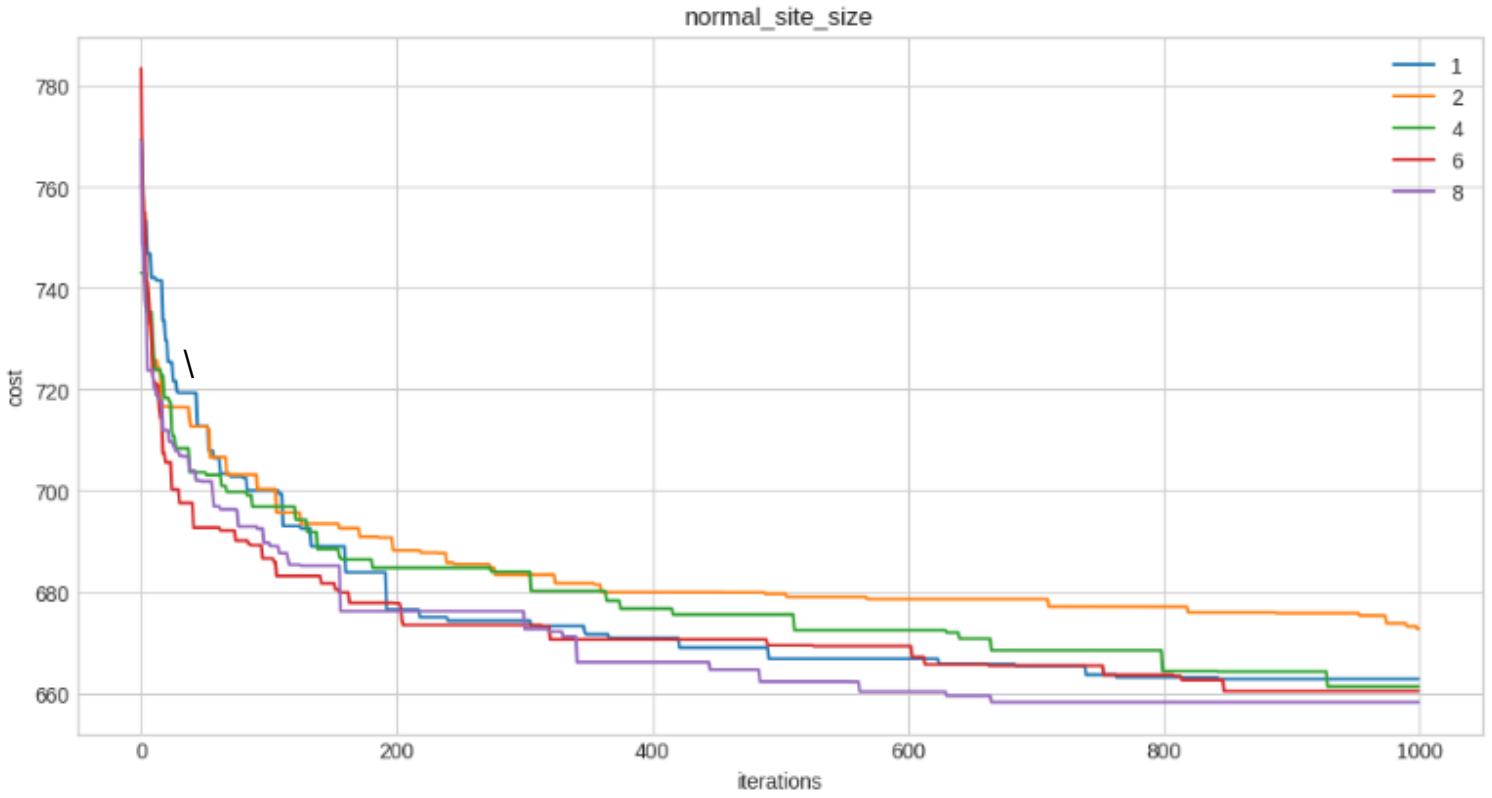
Wartość parametru	Minimalny koszt
2	664.87
4	657.18
8	657.33
12	653.01
16	646.81



5.9. Testy parametru normal_site_size

Parametr ten opisuje ilość sąsiednich rozwiązań generowanych dla każdego normalngo rozwiązania
Testy przeprowadzono dla wartości [2, 4, 8, 12, 16]

Wartość parametru	Minimalny koszt
2	662.77
4	672.76
8	661.29
12	660.39
16	658.22



6. Wnioski

Utworzony przez nas algorytm osiąga bardzo zadowalające wyniki, robiąc to również w przystępnym czasie. Algorytm jest również łatwy w implementacji i zrozumieniu w szczególności gdy porównamy go do deterministycznych alternatyw głównie opierających się na układach równań i nierówności liniowych.

Podczas pracy nad projektem napotkaliśmy jeden istotny problem, a było to sprawdzanie poprawności wyników, ponieważ mamy do czynienia z trudnym problemem optymalizacyjnym. Dla każdego z problemów nie znamy deterministycznego algorytmu i wykonanie ręcznej weryfikacji poprawności jest możliwe jedynie w przypadku małych problemów testowych.

Podsumowując, projekt możemy uznać za udany jako, że w sensownym czasie potrafi znaleźć poprawne rozwiązanie dla zadanego problemu.

7. Podział prac

Szymon Budziak: 33.3 %
Alicja Hurbol: 33.3%
Jakub Szymczak: 33.3 %