Aplikacja do przydzielania akademików studentom - sprawozdanie końcowe

Adam Palka, Bartosz Pajor, Kamil Nowobilski

Spis treści

1. Podział pracy

Etap	Adam Palka	Bartosz Pajor	Kamil Nowobilski
Model zagadnienia	Model matematyczny: funkcja celu, ograniczenia [30%]	Krótki opis modelu zagadnienia, model matematyczny: struktury danych [40%]	Model matematyczny: postać rozwiązania, wejście [30%]
Algorytm opracowanie	definicje sąsiedztwa, lista tabu	liczenie odległości, rozwiązanie	funkcja celu, opracowanie GUI

	[30%]	początkowe, priorytet ze wzgl. na niepełnosprawność [30%]	[40%]
Implementacja aplikacji	calculate_distances() objective_func() generate_neighbourhood() tabu_search() 205/228 (w pliku tabu_search.py) 187/242 (main.py)	generate_data() 39/39 (rand.py) generate_new_data() oraz rysowanie wykresów 53/242 (main.py) starting_solution() 23/228 (tabu_search.py)	GUI (aplikacja.py)
Testy	Opracowanie testów I zestawu danych (wykresy 1 - 24), podsumowanie testów [40%]	Opracowanie testów II zestawu danych (wykresy 25 - 48) [30%]	Opracowanie testów III zestawu danych (wykresy 49 - 64) [30%]
Dokumentacja	Rozdziały: 2.3, 2.5	Rozdziały: 1, 2.1, 2.2, 3	Rozdziały: 2.4

2. Sprawozdanie

2.1 Model zagadnienia - krótki opis

Problem optymalnego przydzielania akademików polega na przypisaniu każdego studenta do jednego z dostępnych akademików, tak aby zminimalizować wartość funkcji celu. Przy przypisywaniu uwzględniane są różnorodne kryteria:

- **Priorytety studentów** dotyczące preferowanych akademików.
- **Odległość między akademikiem, a wydziałem, do którego student należy**.
- **Stopień niepełnosprawności studenta**, który wpływa na jego priorytet w funkcji celu.

- **Pojemność akademików**, ograniczająca liczbę studentów przypisanych do jednego akademika.

Algorytm wykorzystuje podejście oparte na wyszukiwaniu tabu (Tabu Search), które pozwala na eksplorację przestrzeni rozwiązań, unikając zapętlania w lokalnych minimach. Rozwiązanie początkowe generowane jest zgodnie z preferencjami studentów, a następnie ulepszane poprzez analizę sąsiedztwa i aktualizację rozwiązania według określonych kryteriów.

2.2 Model zagadnienia - model matematyczny

2.2.1 Struktury danych

Zmienne dotyczące studenta:

N – liczba studentów $(N \in \boldsymbol{Z}_+)$

r_i – rok studiów studenta i $(r_i \in \boldsymbol{Z}_+)$

T_i – określenie niepełnosprawności studenta i $(T_i \in \{1,2,3,0\})$

X_i – tablica priorytetu akademików studenta i

x_{Wi}, y_{Wi} – położenie wydziału, na którym studiuje student i $(x_{Wi}, y_{Wi} \in \boldsymbol{R})$

Zmienne dotyczące akademików:

M – liczba akademików $(M \in \boldsymbol{Z}_+)$

p_{Aj} – pojemność j-tego akademika $\left(p_{Aj} \in \boldsymbol{Z}_+\right)$

x_{Aj}, y_{Aj} – położenie akademika j $\left(x_{Aj}, y_{Aj} \in \boldsymbol{R}\right)$

$d_{ij} = \sqrt{(x_{Wi} - x_{Ai})^2 + \left(y_{Wi} - x_{Aj}\right)^2}$ – odległość z wydziału studenta i do akademika j $\left(d_{ij} \in \boldsymbol{R}_+\right)$

2.2.2 Postać rozwiązania

Rozwiązaniem jest wektor π, który przypisuje każdemu studentowi i odpowiedni akademik π_i minimalizując funkcję celu.

π – wektor akademików $\{\pi_1, \pi_2, \pi_3, \dots, \pi_N\}$

π_i – akademik i – tego studenta ($\pi_i \in \{1,2,\dots,M\}$)

2.2.3 Wejście

Każdy student podaje listę priorytetu akademików w których chce mieszkać:

$$\forall i \in \{1,2,\dots,N\}, \qquad X_i = \{A_{i1}, A_{i2}, A_{i3}, \dots, A_{iM}\}$$

2.2.4 Funkcja celu

Minimalizacja sumy dystansów z akademika na wydział każdego studenta z uwzględnieniem roku studiów. Im student jest na niższym roku, tym bardziej dbamy o jego komfort – dystans z akademika na wydział powinien być krótszy.

$$\min Z = \sum_{i=1}^{N} \left(\frac{P_i}{100} \left(1 - \frac{r_i}{10}\right) d_{i\pi_i} + \alpha x_{i\pi_i} \right)$$

gdzie:

P_i – priorytet wyznaczany na podstawie stopnia niepełnosprawności

$$P(T_i) = \begin{cases} 100, & T_i = 1 \\ 75, & T_i = 2 \\ 50, & T_i = 3 \\ 25, & T_i = 0 \end{cases}$$

$x_{i\pi_i}$ – pozycja akademika π na liście X_i

α – współczynnik regulujący wpływ preferencji studenta.

2.2.5 Ograniczenia

Każdy akademik A_j ma ograniczoną pojemność p_{Aj}:

$$\forall j \in \{1,2,\dots,M\}, \qquad \sum_{i=1}^{N} g(\pi_i, j) \leq p_{Aj}$$

gdzie $g(\pi_i, j)$ to funkcja zwracająca 1, gdy $\pi_i == j$. W przeciwnym razie zwraca 0.

Liczba miejsc w akademikach jest większa niż liczba wszystkich studentów:

$$\sum_{j=1}^{M} p_{Aj} > N$$

2.3 Algorytm

Algorytm został zrealizowany za pomocą głównej funkcji *tabu_search* oraz 4 funkcji pomocniczych wywoływanych wewnątrz głównej funkcji.

1. *tabu_search*

Implementuje algorytm Tabu Search, aby znaleźć optymalne przypisanie studentów do akademików.

Wejście:

- start_solution: Rozwiązanie początkowe.
- years: Rok studiów każdego studenta.
- disabilities: Stopień niepełnosprawności każdego studenta.
- prior_list: Lista priorytetów każdego studenta.
- students_sex: Płeć każdego studenta.
- departments: Lista wydziałów każdego studenta.
- dorm_capacity: Pojemność każdego akademika.
- dorm_pos: Współrzędne akademików.
- dep_pos: Współrzędne wydziałów.
- neighbourhood_type: Typ generowanego sąsiedztwa.
- max_iterations: Maksymalna liczba iteracji algorytmu.
- tabu_list_size: Rozmiar listy tabu.
- alpha: Współczynnik wag dla funkcji celu.

Wyjście:

- Najlepsze przypisanie studentów.
- Wartość funkcji celu dla najlepszego przypisania.
- Iteracje i wartości funkcji celu dla analizy.

Pseudokod funkcji:

```
FUNKCJA tabu_search(
    start_solution, years, disabilities, prior_list,
    students_sex, departments, dorm_capacity,
    dorm_pos, dep_pos, neighbourhood_type,
    max_iterations, tabu_list_size, alpha
):
    # Oblicz macierz odległości
    distances = calculate_distances(dorm_pos, dep_pos)

    # Inicjalizacja zmiennych
    current_solution = start_solution
    best_solution = current_solution
    best_objective = objective_func(
        current_solution, years, disabilities, prior_list, departments, distances,
alpha
    )

    tabu_list = []
    iteration = 0
    no_improvement_counter = 0

    # Do śledzenia wyników w trakcie algorytmu
    iterations = []
    objectives = []

    WHILE iteration < max_iterations:

        # Zbieranie danych dla analiz
        iterations.APPEND(iteration)
        objectives.APPEND(best_objective)

        # Generowanie sąsiedztwa dla bieżącego rozwiązania
        neighbourhood = generate_neighbourhood(current_solution, prior_list,
dorm_capacity, neighbourhood_type)
```

```
        # Sprawdzenie, czy są dostępne sąsiedzi
        JEŚLI neighbourhood jest puste:
            PRZERWIJ

        # Filtracja sąsiadów na podstawie tabu_list
        filtered_neighbourhood = []
        DLA solution W neighbourhood:
            JEŚLI solution NIE JEST w tabu_list:
                filtered_neighbourhood.APPEND(solution)

        JEŚLI filtered_neighbourhood jest puste:
            PRZERWIJ

        # Inicjalizacja wartości dla najlepszego sąsiada w bieżącej iteracji
        best_neighbour = filtered_neighbourhood[0]
        best_neighbour_objective = objective_func(
            best_neighbour, years, disabilities, prior_list, departments, distances,
alpha
        )

        # Iteracja po sąsiedztwie w celu znalezienia najlepszego sąsiada
        DLA neighbour W filtered_neighbourhood:
            neighbour_objective = objective_func(
                neighbour, years, disabilities, prior_list, departments, distances,
alpha
            )

            # Kryterium aspiracji
            JEŚLI (neighbour_objective < best_neighbour_objective) LUB
                 (neighbour W tabu_list I neighbour_objective < best_objective):
                best_neighbour = neighbour
                best_neighbour_objective = neighbour_objective

        # Aktualizacja najlepszego rozwiązania globalnie
        JEŚLI best_neighbour_objective < best_objective:
            best_solution = best_neighbour
            best_objective = best_neighbour_objective
            no_improvement_counter = 0
        INACZEJ:
            no_improvement_counter += 1

        # Aktualizacja bieżącego rozwiązania
        current_solution = best_neighbour
```

```
        # Dodanie rozwiązania do listy tabu
        tabu_list.APPEND(current_solution)

        # Ograniczenie rozmiaru listy tabu
        JEŚLI len(tabu_list) > tabu_list_size:
            DELETE najstarszy_element_z(tabu_list)

        # Sprawdzenie kryterium końca (10 iteracji bez poprawy)
        JEŚLI no_improvement_counter >= 10:
            PRZERWIJ

        # Zwiększenie liczby iteracji
        iteration += 1

    # Zwrócenie najlepszego rozwiązania oraz wartości funkcji celu
    ZWRÓĆ best_solution, best_objective, iterations, objectives
```

2. *calculate_distances*

Oblicza odległości między wszystkimi akademikami a wydziałami na podstawie współrzędnych.

Wejście:

- derm_pos: Lista współrzędnych akademików (x, y).
- dep_pos: Lista współrzędnych wydziałów (x, y).

Wyjście:

- Dwuwymiarowa macierz odległości między wydziałami, a akademikami.

Pseudokod funkcji:

```
FUNKCJA calculate_distances(dorm_pos, dep_pos):
    macierz_odległości = []
    DLA akademik W dorm_pos:
        lista_odległości = []
        DLA wydział W dep_pos:
            odległość = sqrt((akademik.x - wydział.x)^2 + (akademik.y -
wydział.y)^2)
            DODAJ odległość DO lista_odległości
        DODAJ lista_odległości DO macierz_odległości
```

```
ZWRÓĆ macierz_odległości
```

3. *starting_solution*

Generuje początkowe rozwiązanie, przypisując studentów do akademików w sposób zgodny z ich priorytetami i dostępnymi miejscami.

Wejście:

- prior_list: Lista priorytetów każdego studenta.
- disabilities: Stopień niepełnosprawności każdego studenta.
- students_sex: Płeć każdego studenta.
- dorm_capacity: Pojemność poszczególnych akademików.

Wyjście:

- Lista przypisania każdego studenta do akademika (bądź *None* jeśli student nie został przypisany)

Pseudokod funkcji:

```
FUNKCJA starting_solution(prior_list, disabilities, students_sex,
dorm_capacity):
    wynik = [None] * długość(prior_list)
    pojemność = [0] * długość(dorm_capacity)
    posortowani_studenci = sortuj_index(prior_list, klucz = -disabilities[student])

    DLA student W posortowani_studenci:
        DLA akademik W prior_list[student]:
            JEŚLI pojemność[akademik] < dorm_capacity[akademik]:
                wynik[student] = akademik
                pojemność[akademik] += 1
                PRZERWIJ

    ZWRÓĆ wynik
```

4. *objective_func*

Oblicza wartość funkcji celu, oceniając jakość danego rozwiązania. Funkcja celu uwzględnia odległości między akademikami a wydziałami oraz inne czynniki, np. stopień niepełnosprawności.

Wejście:

- input_vector: Aktualne przypisanie studentów do akademików.
- years: Rok studiów każdego studenta.
- disabilities: Stopień niepełnosprawności każdego studenta.
- prior_list: Lista priorytetów każdego studenta.
- departments: Lista wydziałów każdego studenta.
- distances: Macierz odległości między akademikami, a wydziałami.
- alpha: Współczynnik wag do funkcji celu.

Wyjście:

- Wartość funkcji celu dla danego rozwiązania.

Pseudokod funkcji:

```
FUNKCJA objective_func(input_vector, years, disabilities, prior_list,
departments, distances, alpha):
    wynik = 0
    DLA i od 0 do długość(input_vector):
        JEŚLI input_vector[i] != None:
            akademik = input_vector[i]
            wydział = departments[i]
            priorytet_ranking = pozycja akademika w prior_list[i]
            wynik += (priorytet(dla disabilities) * odległość) + alpha *
priorytet_ranking
    ZWRÓĆ round(wynik, 3)
```

5. *generate_neighbourhood*

Tworzy zestaw możliwych rozwiązań (sąsiedztwo) przez modyfikację bieżącego przypisania studentów.

Wejście:

- current_solution: Obecne przypisanie studentów do akademików.
- prior_list: Lista priorytetów każdego studenta.
- dorm_capacity: Pojemność poszczególnych akademików.
- neighbourhood_type: Rodzaj generowane sąsiedztwa.

Wyjście:

- Lista sąsiedztwa.

Pseudokod funkcji:

```
FUNKCJA generate_neighbourhood(current_solution, prior_list,
dorm_capacity, neighbourhood_type):
    sąsiedztwo = []

    JEŚLI neighbourhood_type == "change_dorm":
        DLA każdego studenta:
            SPRAWDŹ zmiany akademików dla studenta

    JEŚLI neighbourhood_type == "swap_students":
        ZAMIEN dwa akademiki między studentami

    JEŚLI neighbourhood_type == "move_group":
        PRZENIEŚ całą grupę z jednego akademika do innego

    ZWRÓĆ sąsiedztwo
```

Rodzaje sąsiedztwa:

- *change_dorm*: Zmiana akademika dla studenta.
- *swap_students*: Zamiana dwóch studentów ze sobą.
- *move_group*: Przeniesienie całej grupy studentów.
- *both*: Kombinacja powyższych.

2.4 Aplikacja

Wymagania dotyczące uruchomienia aplikacji

Aplikacja została stworzona z myślą o wszechstronnym wykorzystaniu i jest zgodna z systemami operacyjnymi Windows, Linux oraz MacOS. Kod źródłowy aplikacji został napisany w języku Python, co pozwala na łatwe rozwijanie oraz dostosowywanie jej funkcji w przyszłości. Aby aplikacja działała poprawnie, konieczne jest posiadanie zainstalowanej wersji Pythona 3.8 lub nowszej. Aplikacja również jest dostępna poprzez otworzenie pliku .exe, co umożliwia jej uruchomienie na systemach Windows bez potrzeby instalowania Pythona. Dzięki temu użytkownicy mogą korzystać z programu bez obawy o konfigurację środowiska, co upraszcza proces użytkowania i zapewnia szeroką dostępność.

Przed uruchomieniem w vsc należy upewnić się, że wszystkie wymagane biblioteki są zainstalowane. Lista zależności znajduje się w pliku requirements.txt. Główne biblioteki wykorzystywane przez aplikację to NumPy, Matplotlib i Pandas, które odpowiadają za przetwarzanie danych, tworzenie wykresów oraz analizę wyników.

Format danych wejściowych i wyjściowych

Aplikacja obsługuje różne formaty danych, co czyni ją elastyczną i łatwą do integracji z innymi narzędziami. Dane wejściowe mogą być dostarczone w formie plików tekstowych (.txt), arkuszy kalkulacyjnych w formacie .csv, jeśli aplikacja jest wykorzystywana w trybie API. Użytkownik ma również możliwość wprowadzania danych bezpośrednio w terminalu lub za pośrednictwem interfejsu użytkownika.

Wyniki przetwarzania są generowane w podobnych formatach – .txt, lub jako pliki graficzne (.png) w przypadku wizualizacji. Na przykład, aplikacja może wygenerować tabelę wyników z dodatkowymi kolumnami prezentującymi przetworzone dane, np. wartości przefiltrowane lub znormalizowane.

Przykładowy plik wyjściowy:

Funkcjonalność aplikacji

Aplikacja została zaprojektowana z myślą o wieloetapowym przetwarzaniu danych oraz wizualizacji wyników. Jej funkcjonalność można podzielić na kilka kluczowych obszarów:

1. **Import danych:**
   Aplikacja umożliwia użytkownikowi łatwe wczytanie danych wejściowych z plików tekstowych, arkuszy kalkulacyjnych. Można zaimportować dane pochodzące z pliku tekstowego, arkusza kalkulacyjnego lub wpisać bezpośrednio dane w aplikacji.
2. **Przetwarzanie danych:**
   Aplikacja wykonuje szeroki zakres operacji matematycznych oraz analitycznych, co jest opisane w części dotyczącej algorytmu.
3. **Wizualizacja wyników:**
   Aplikacja generuje wykresy wartości funkcji celu w kolejnych iteracjach.

   Dzięki temu użytkownik może w prosty sposób zrozumieć dane oraz zidentyfikować kluczowe wzorce i zależności. Wygenerowane wykresy są zapisywane w formatach graficznych, które można łatwo wykorzystać w raportach lub prezentacjach. Każde uruchomienie algorytmu zapisuje się i jest zapamiętywane przez program.
4. **Eksport wyników:**
   Przetworzone dane oraz wykresy mogą być zapisane w plikach wyjściowych w wybranym formacie, takim jak CSV, JSON czy PNG. Dzięki temu użytkownik

może łatwo wykorzystać wyniki w innych narzędziach, np. arkuszach kalkulacyjnych, raportach czy aplikacjach do analizy danych.

Aplikacja zawiera również funkcję umożliwiającą szybki dostęp do wyników ostatnich działań algorytmu za pomocą zakładki „Ostatnie działanie algorytmu". Dzięki temu użytkownicy nie muszą ponownie uruchamiać całego procesu obliczeniowego, co pozwala na zaoszczędzenie czasu. Zamiast tego, mogą odwołać się do wyników poprzednich obliczeń, co jest szczególnie przydatne, gdy algorytm jest czasochłonny lub obliczenia muszą być wielokrotnie powtarzane na tych samych danych.

Po wywołaniu zakładki, aplikacja pokazuje historię ostatnich obliczeń i wyniki, umożliwiając użytkownikowi wybór konkretnego działania, które może zostać ponownie załadowane i wykorzystane. Funkcja ta zapewnia elastyczność i wygodę, umożliwiając łatwiejsze analizowanie wyników w kontekście wcześniejszych działań bez potrzeby przeprowadzania pełnych obliczeń od podstaw. To rozwiązanie zwiększa efektywność pracy i upraszcza proces analizy, szczególnie w przypadku rozbudowanych procesów obliczeniowych.

2.5 Testy

Testy przeprowadzono dla różnych zestawów danych wejściowych, badając zbieżność algorytmu i czas wykonywania obliczeń w zależności od liczby studentów, akademików, czy wydziałów oraz parametrów funkcji tabu: maksymalnej liczby iteracji oraz długości listy tabu.

Algorytm wywoływano dla każdego rodzaju zdefiniowanego wcześniej sąsiedztwa:

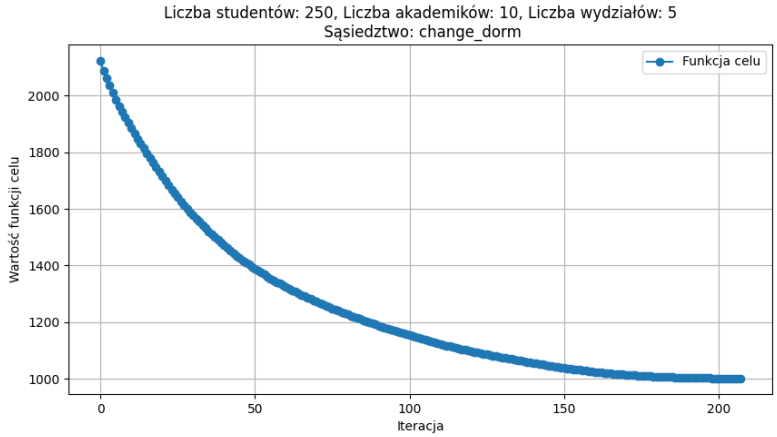
- *change dorm*
- *swap students*
- *move group*
- *both*

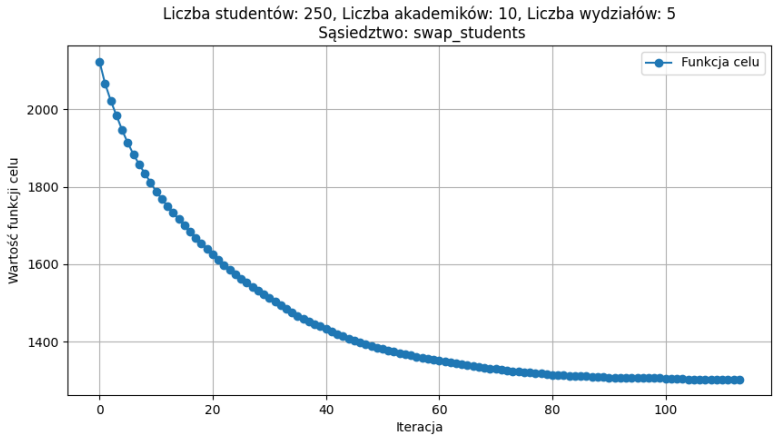
Pierwszy zestaw parametrów wejściowych:

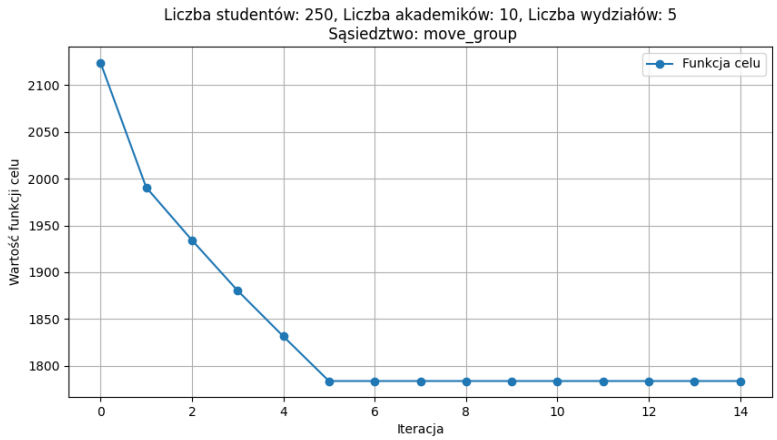
Lp.	Liczba studentów	Liczba akademików	Liczba wydziałów	*max_iterations*	*tabu_list_size*
1.	250	10	5	1000	100
2.	250	5	5	1000	100
3.	250	2	5	1000	100
4.	250	10	10	1000	100

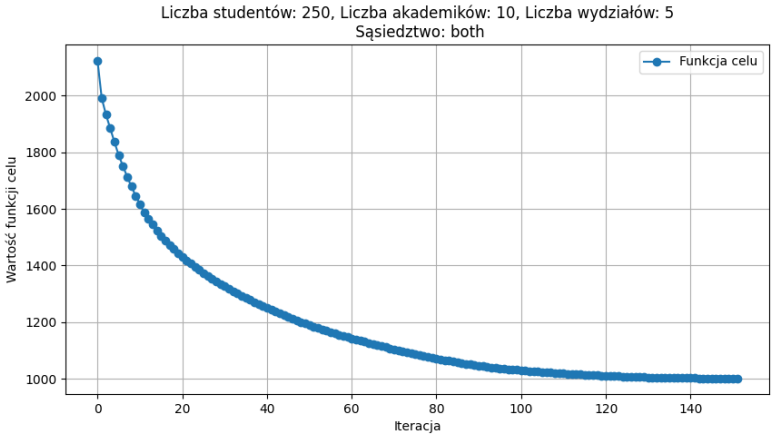
5.	250	5	10	1000	100
6.	250	2	10	1000	100

1) Liczba studentów: 250
   Liczba akademików: 10
   Liczba wydziałów: 5
   max_iterations: 1000
   tabu_list_size: 100



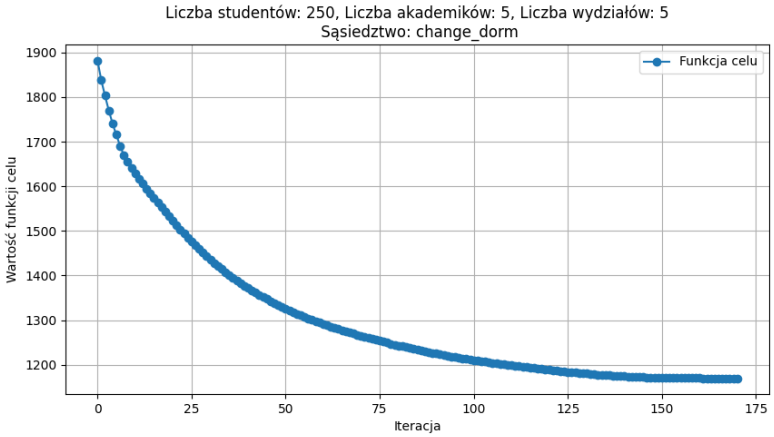


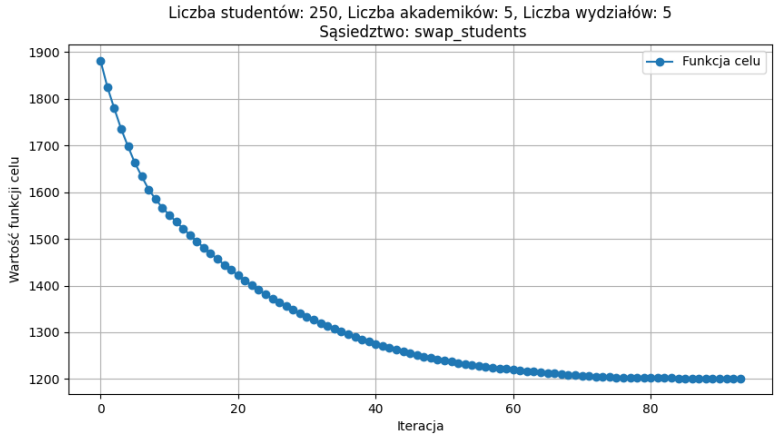


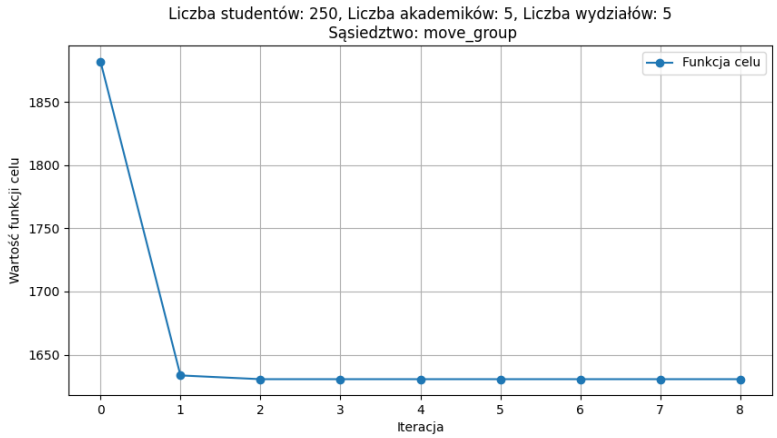


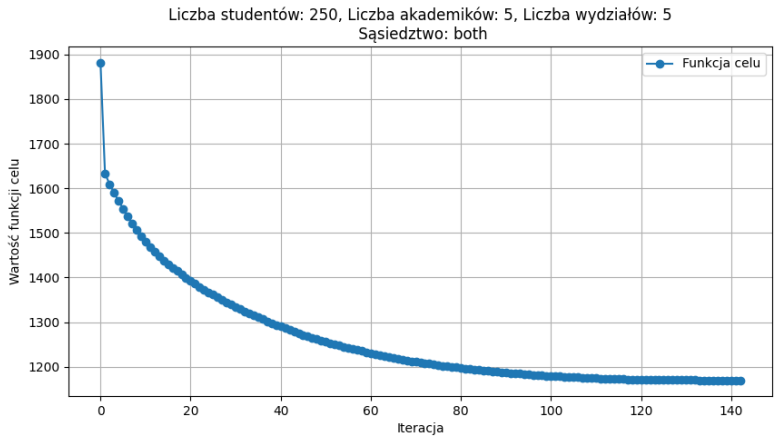
Sąsiedztwo	Wartość początkowa funkcji celu	Wartość końcowa funkcji celu	Czas obliczeń
change dorm	2123.96	1002.50	39.99 [s]
swap students	2123.96	1302.98	289.35 [s]
move group	2123.96	1783.88	0.06 [s]
both	2123.96	1002.5	383.55 [s]

2) Liczba studentów: 250
   Liczba akademików: 5
   Liczba wydziałów: 5
   max_iterations: 1000
   tabu_list_size: 100



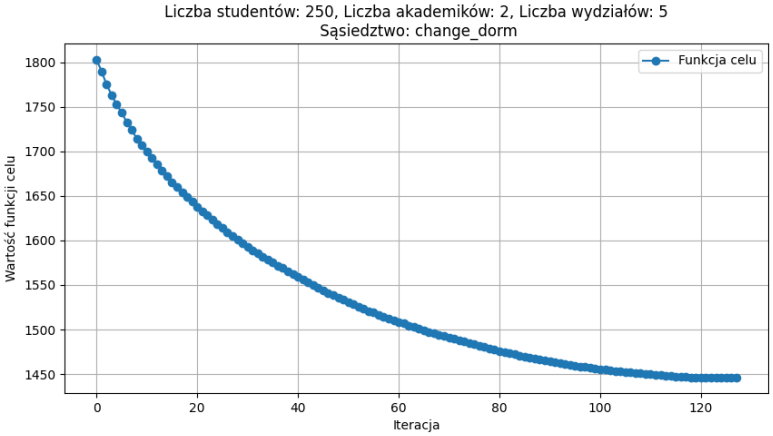


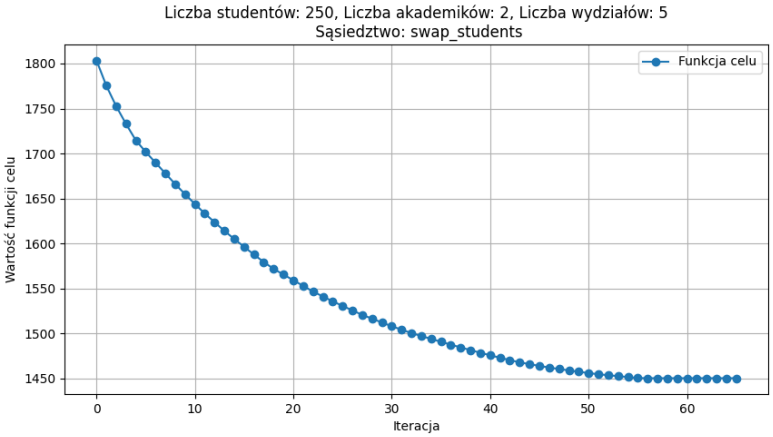


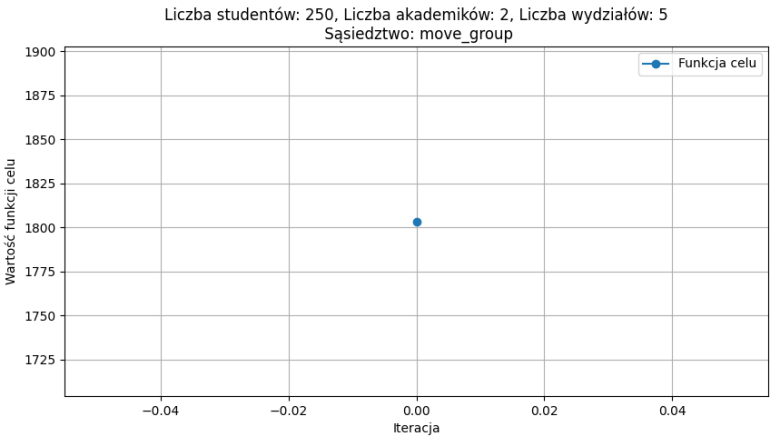


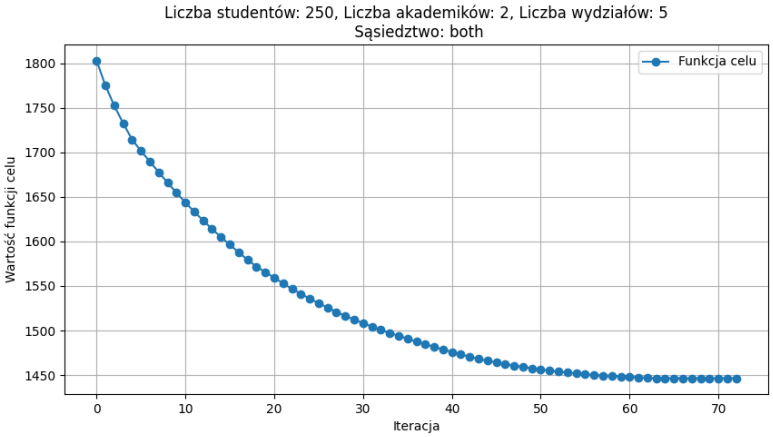
Sąsiedztwo	Wartość początkowa funkcji celu	Wartość końcowa funkcji celu	Czas obliczeń
change dorm	1882.23	1169.94	15.02 [s]
swap students	1882.23	1201.67	215.56 [s]
move group	1882.23	1630.59	0.01 [s]
both	1882.23	1169.94	344.25 [s]

3) Liczba studentów: 250
Liczba akademików: 2
Liczba wydziałów: 5
max_iterations: 1000
tabu_list_size: 100



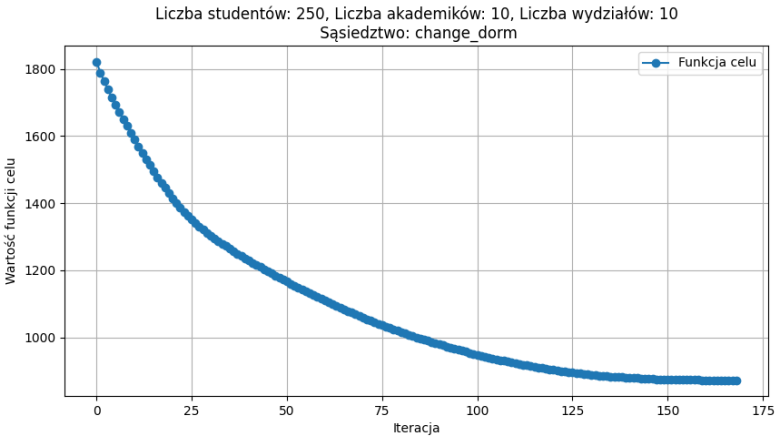


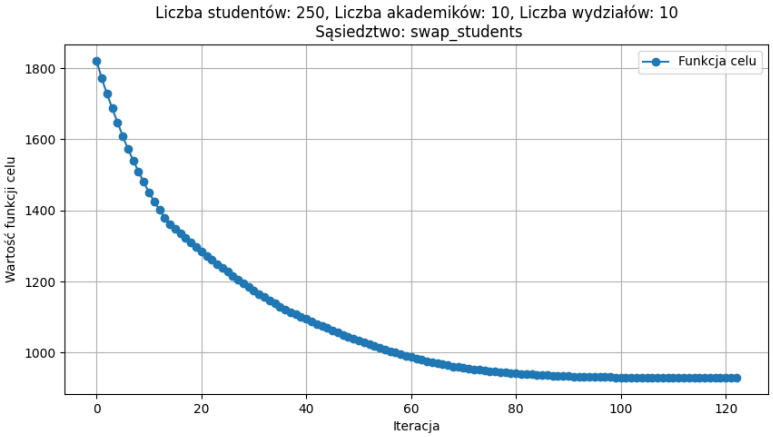


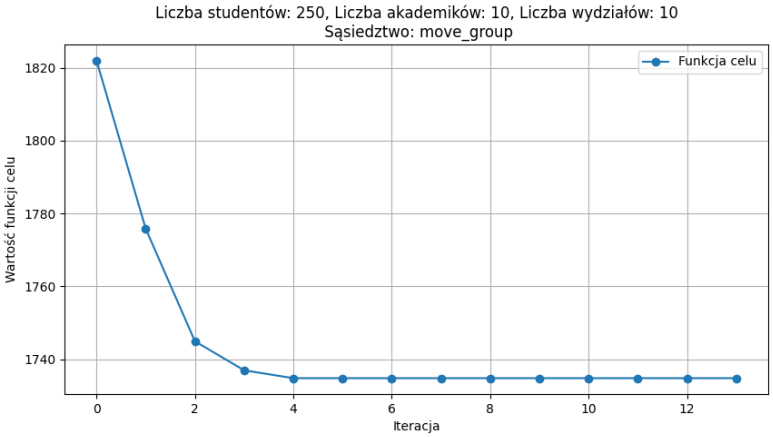


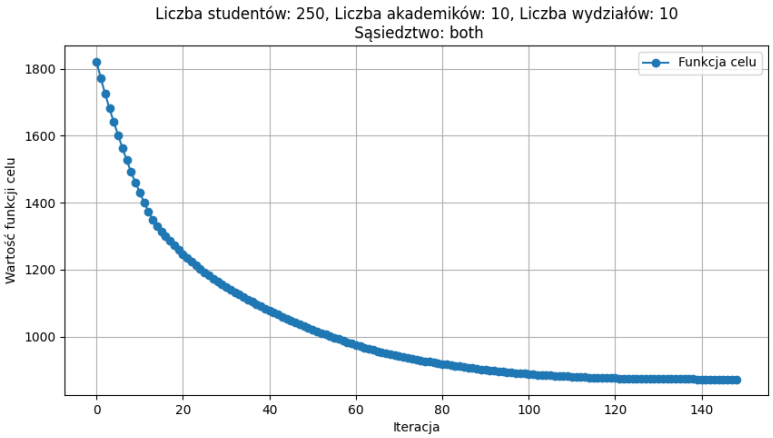
Sąsiedztwo	Wartość początkowa funkcji celu	Wartość końcowa funkcji celu	Czas obliczeń
change dorm	1803.49	1446.45	2.33 [s]
swap students	1803.49	1450.48	99.59 [s]
move group	1803.49	1803.49	0.01 [s]
both	1803.49	1446.45	109.70 [s]

4) Liczba studentów: 250
   Liczba akademików: 10
   Liczba wydziałów: 10
   max_iterations: 1000
   tabu_list_size: 100





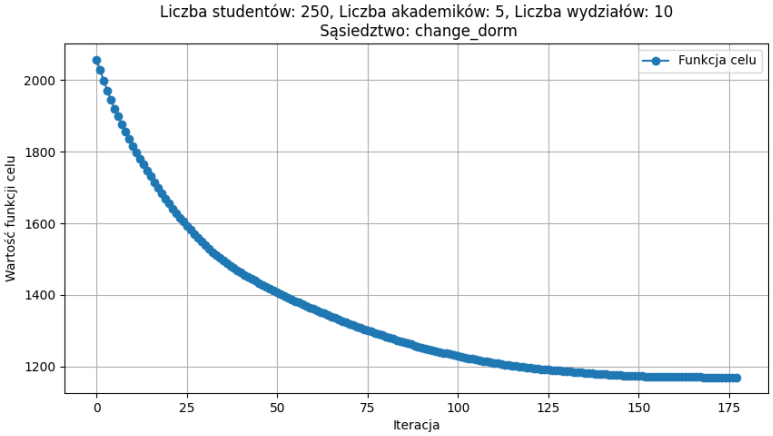


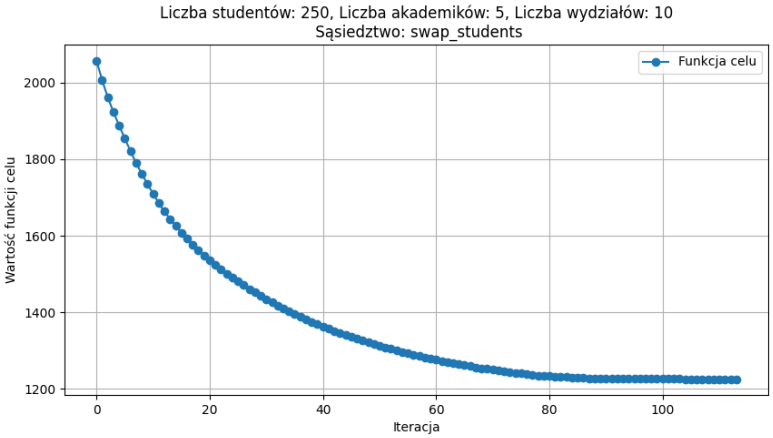


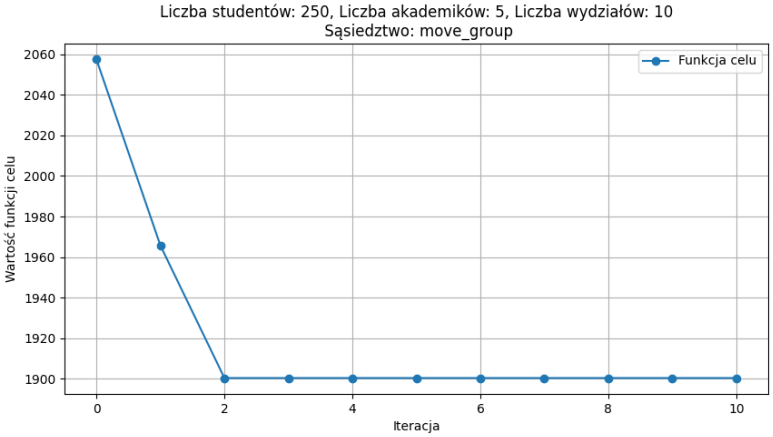
Sąsiedztwo	Wartość początkowa funkcji celu	Wartość końcowa funkcji celu	Czas obliczeń
change dorm	1822.02	873.26	32.68 [s]
swap students	1822.02	928.02	304.25 [s]
move group	1822.02	1734.80	0.06 [s]

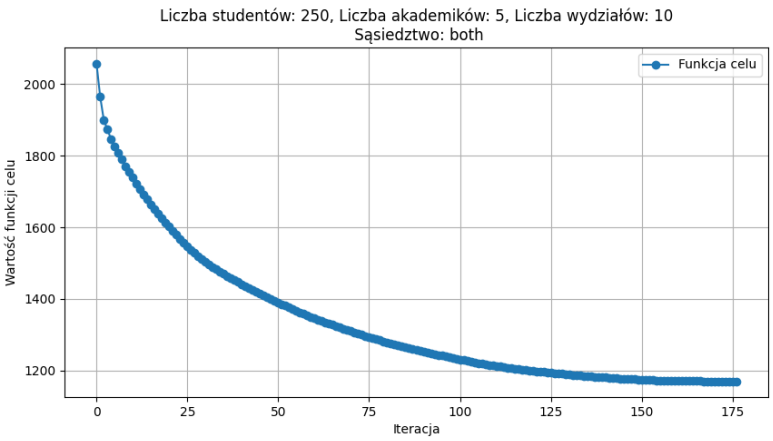
both	1822.02	873.26	395.26 [s]

5) Liczba studentów: 250
Liczba akademików: 5
Liczba wydziałów: 10
max_iterations: 1000
tabu_list_size: 100



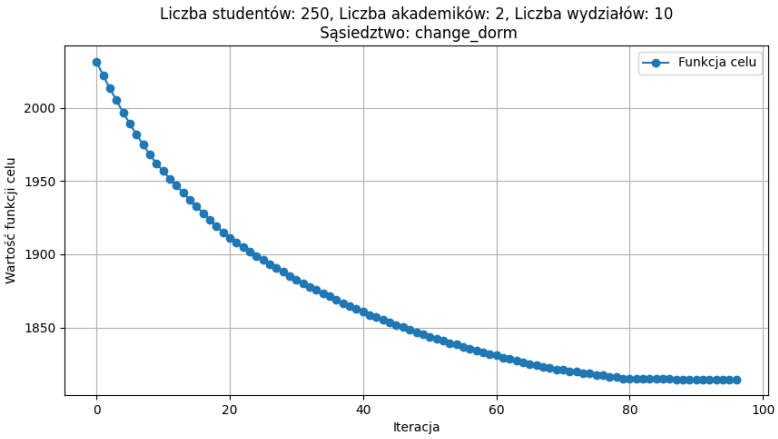


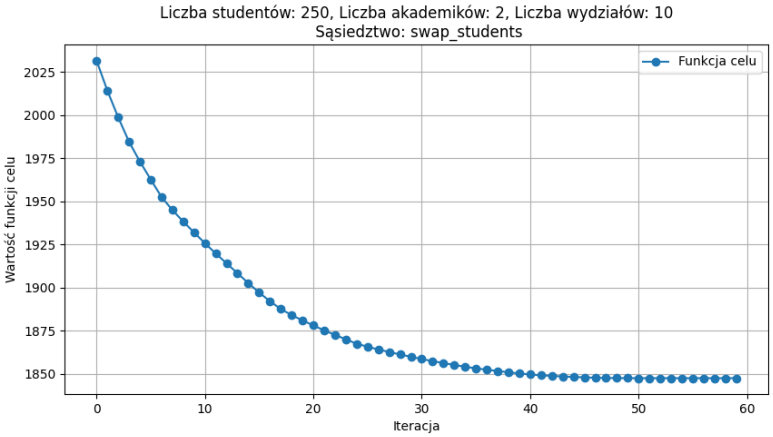


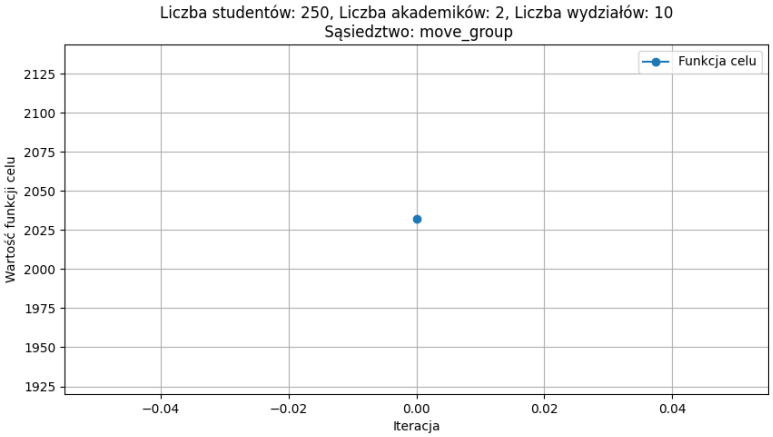


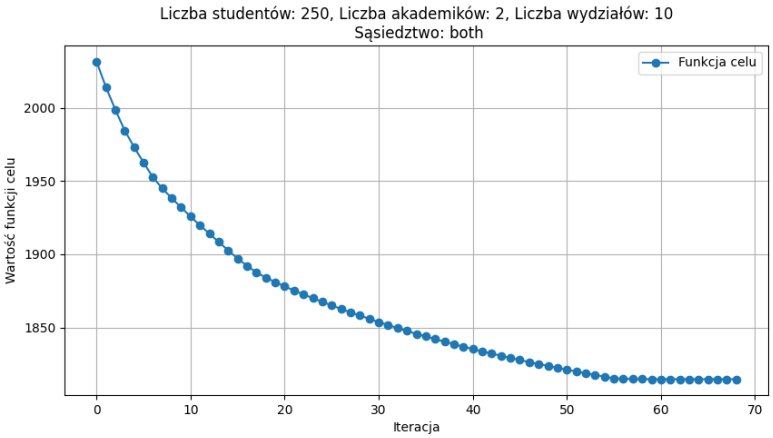
Sąsiedztwo	Wartość początkowa funkcji celu	Wartość końcowa funkcji celu	Czas obliczeń
change dorm	2057.48	1170.43	15.04 [s]
swap students	2057.48	1226.12	256.83 [s]
move group	2057.48	1900.28	0.01 [s]
both	2057.48	1170.43	412.32 [s]

6) Liczba studentów: 250
Liczba akademików: 2
Liczba wydziałów: 10
max_iterations: 1000
tabu_list_size: 100







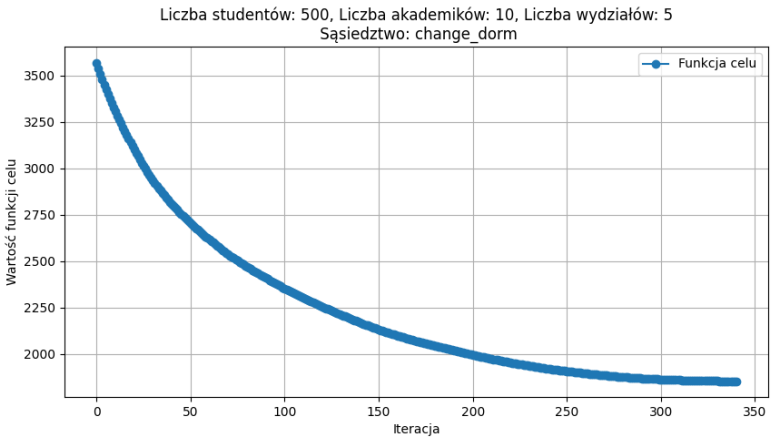


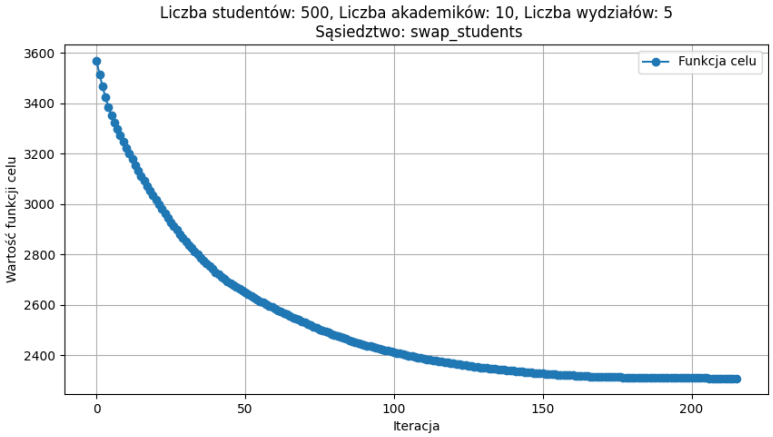
Sąsiedztwo	Wartość początkowa funkcji celu	Wartość końcowa funkcji celu	Czas obliczeń
change dorm	2031.84	1814.69	2.15 [s]
swap students	2031.84	1847.534	105.14 [s]
move group	2031.84	2031.84	0.01 [s]
both	2031.84	1814.69	109.03 [s]

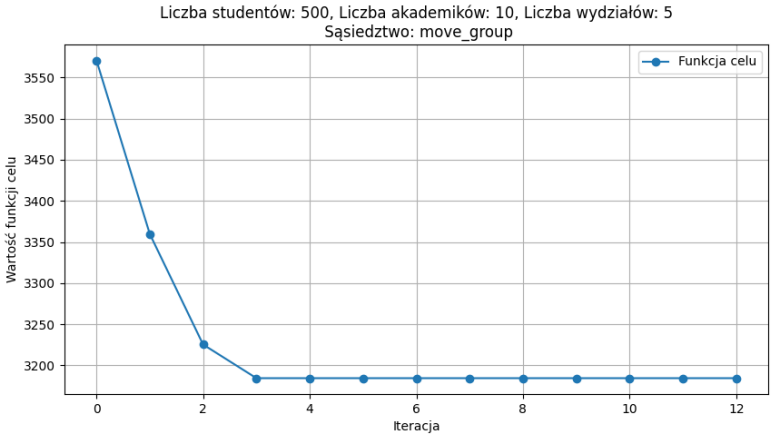
Drugi zestaw parametrów wejściowych:

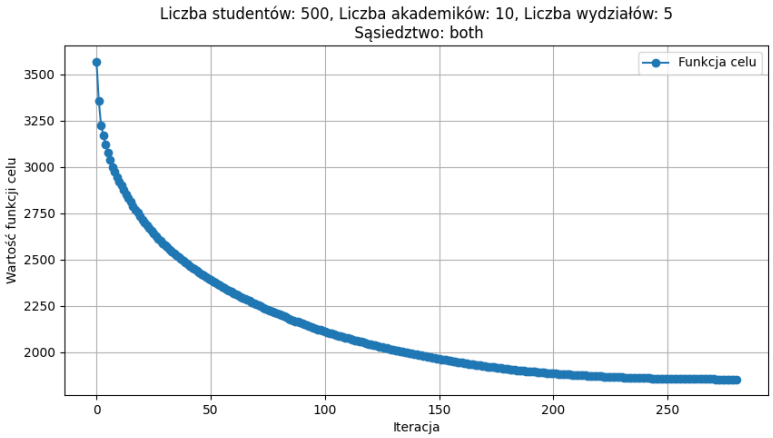
Lp.	Liczba studentów	Liczba akademików	Liczba wydziałów	*max_iterations*	*tabu_list_size*
1.	500	10	5	1000	100
2.	500	5	5	1000	100
3.	500	2	5	1000	100
4.	500	10	10	1000	100
5.	500	5	10	1000	100
6.	500	2	10	1000	100

1) Liczba studentów: 500
   Liczba akademików: 10
   Liczba wydziałów: 5
   max_iterations: 1000
   tabu_list_size: 100



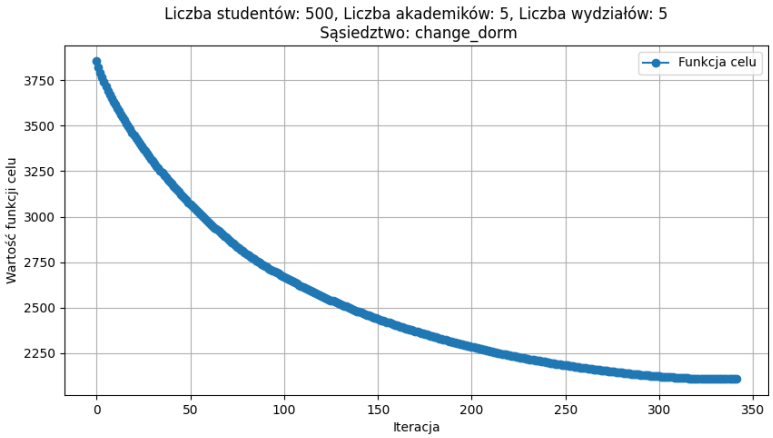


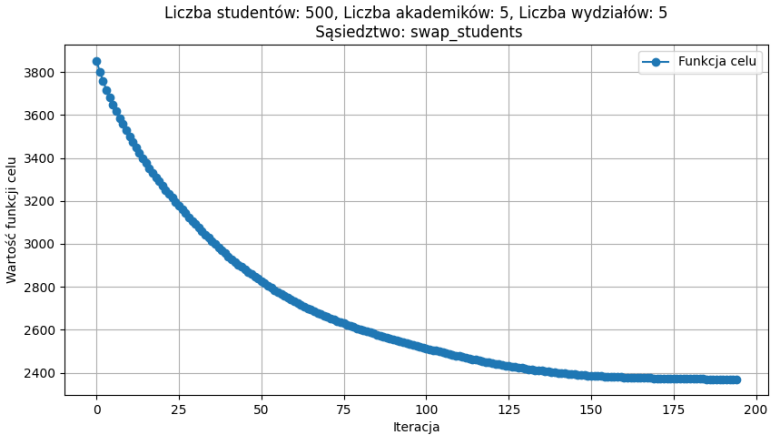


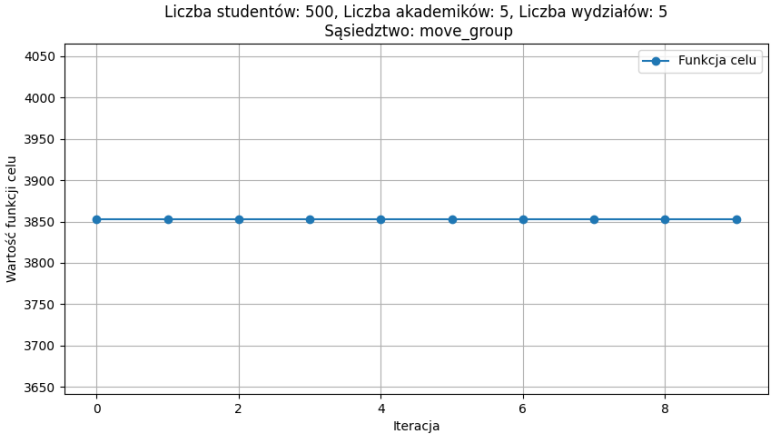


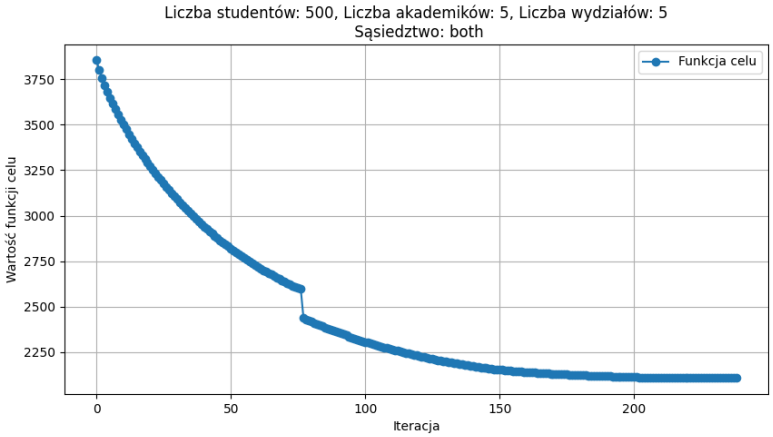
Sąsiedztwo	Wartość początkowa funkcji celu	Wartość końcowa funkcji celu	Czas obliczeń
change dorm	3570.74	1854.86	274.09 [s]
swap students	3570.74	2309.35	4421.39 [s]
move group	3570.74	3184.46	0.12 [s]
both	3570.74	1854.02	5691.37 [s]

2) Liczba studentów: 500
Liczba akademików: 5
Liczba wydziałów: 5
max_iterations: 1000
tabu_list_size: 100



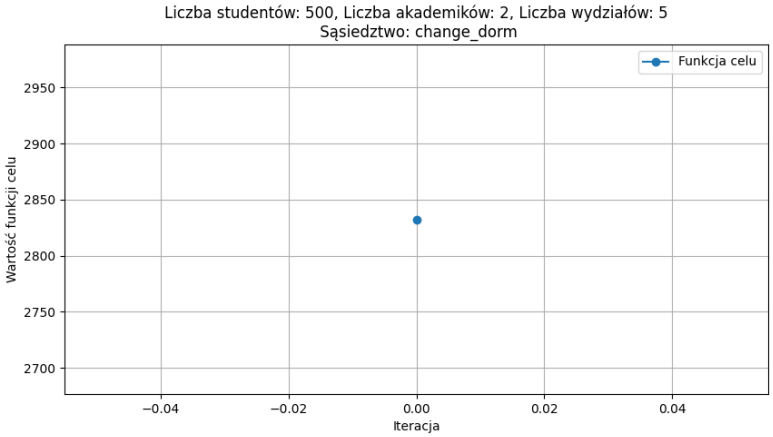


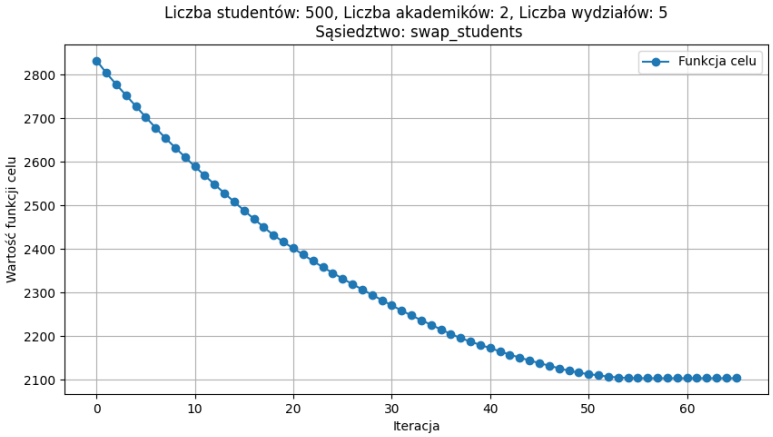


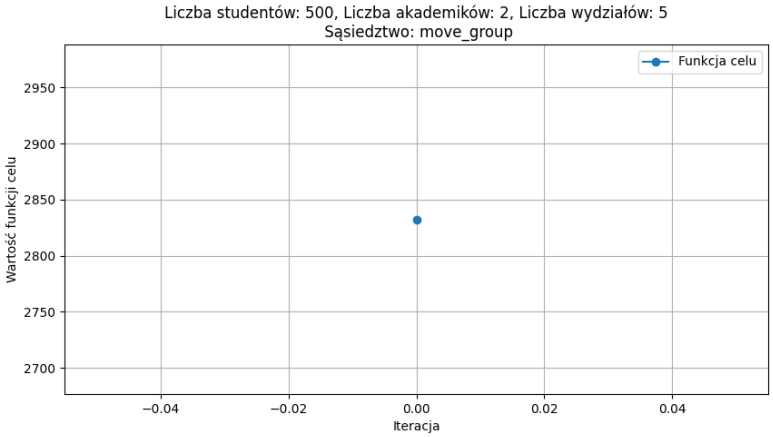


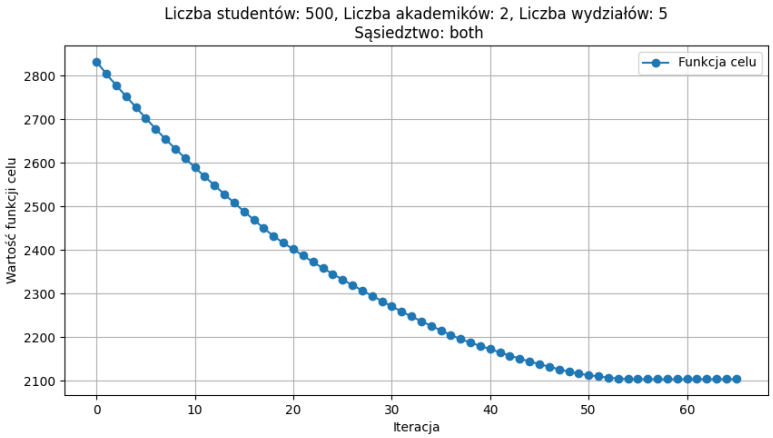
Sąsiedztwo	Wartość początkowa funkcji celu	Wartość końcowa funkcji celu	Czas obliczeń
change dorm	3853.39	2107.83	112.46 [s]
swap students	3853.39	2370.59	3440.31 [s]
move group	3853.39	3853.39	0.02 [s]
both	3853.39	2107.833	4224.02 [s]

3) Liczba studentów: 500
Liczba akademików: 2
Liczba wydziałów: 5
max_iterations: 1000
tabu_list_size: 100





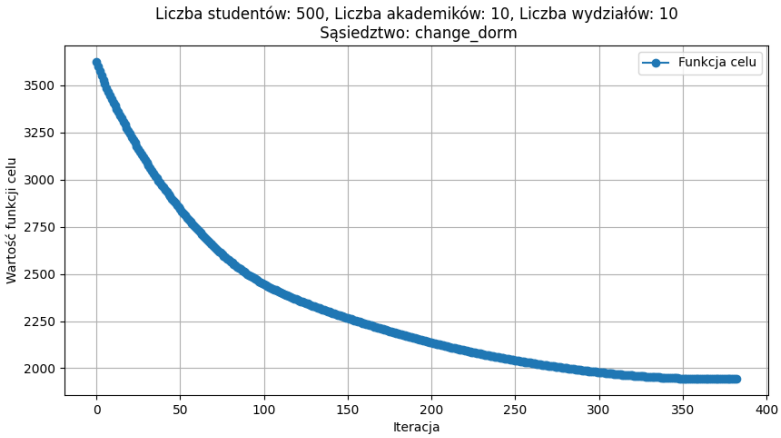


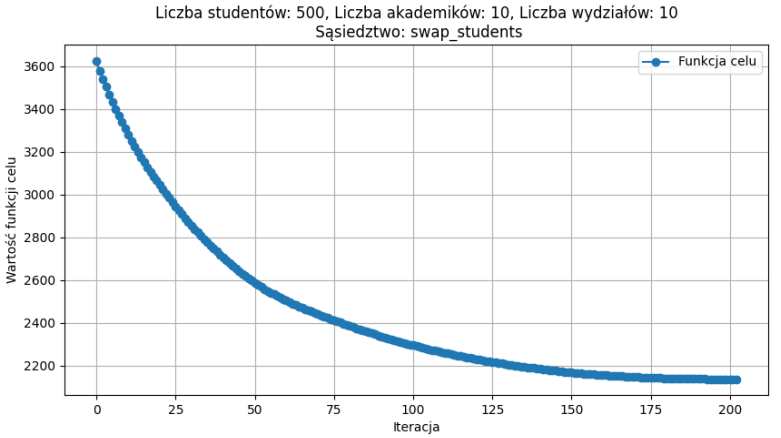


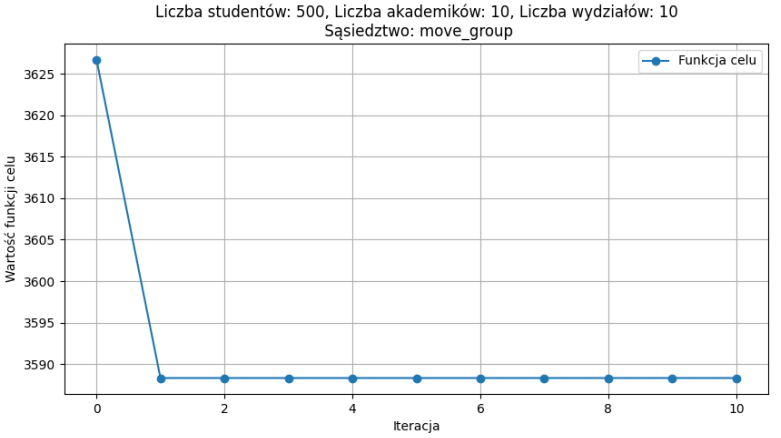
Sąsiedztwo	Wartość początkowa funkcji celu	Wartość końcowa funkcji celu	Czas obliczeń
change dorm	2832.45	2832.45	0.01 [s]
swap students	2832.45	2103.42	103.67 [s]
move group	2832.45	2832.45	0.01 [s]

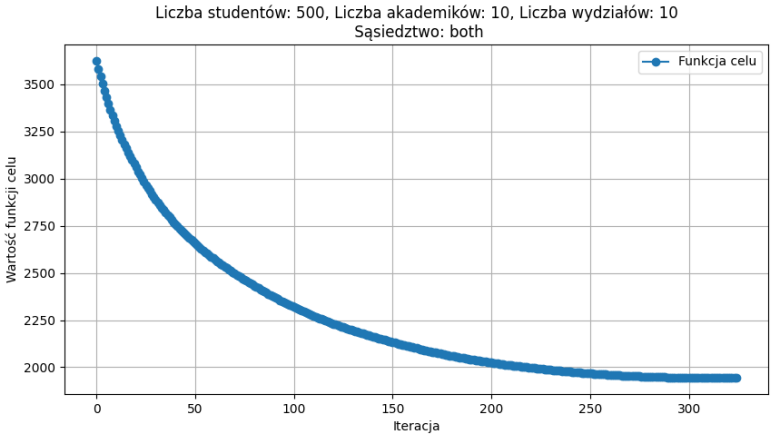
both	2832.45	2103.42	103.93 [s]

4) Liczba studentów: 500
Liczba akademików: 10
Liczba wydziałów: 10
max_iterations: 1000
tabu_list_size: 100



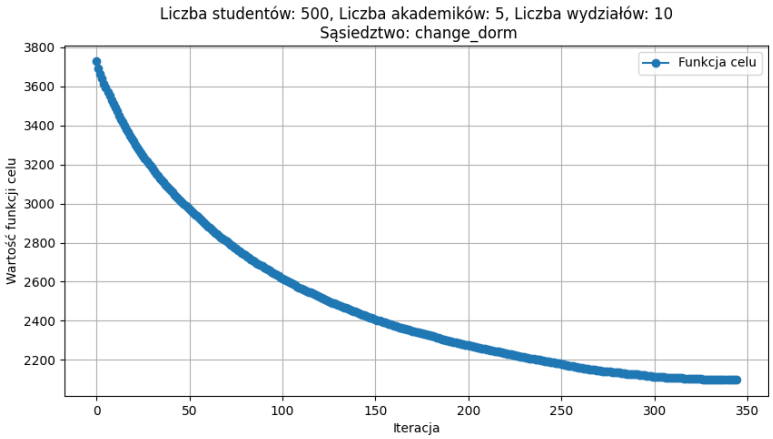


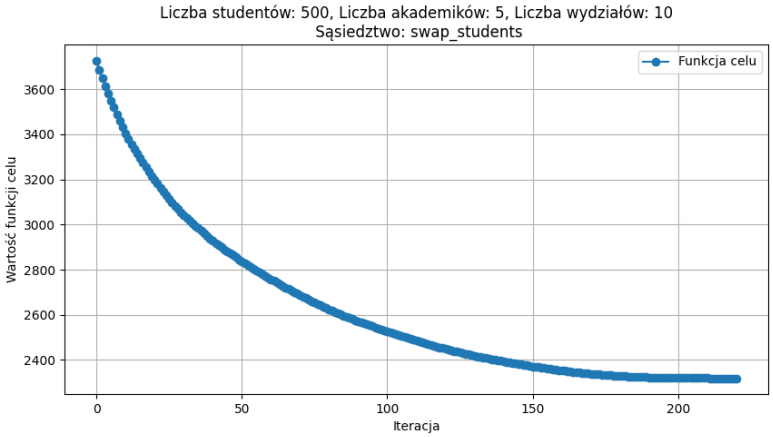


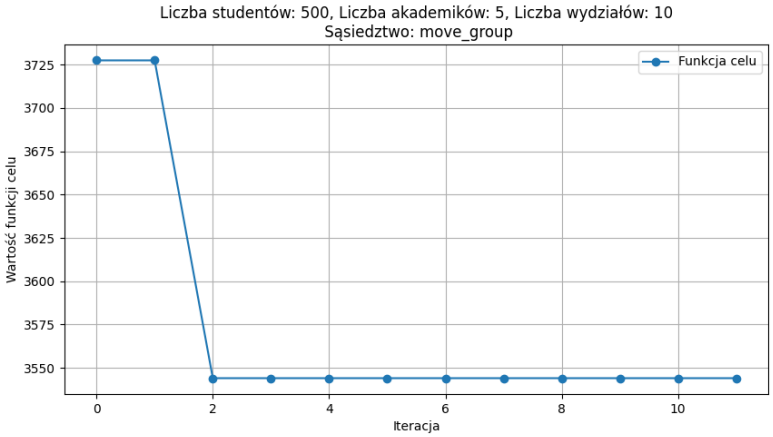


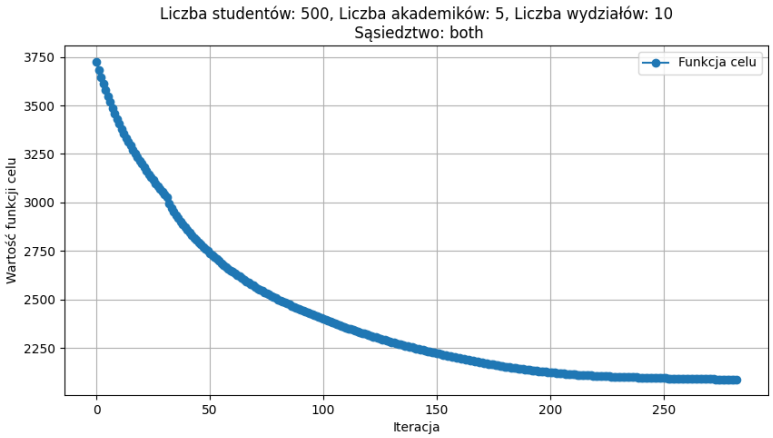
Sąsiedztwo	Wartość początkowa funkcji celu	Wartość końcowa funkcji celu	Czas obliczeń
change dorm	3626.71	1942.03	293.45 [s]
swap students	3626.71	2136.98	4052.02 [s]
move group	3626.71	3588.33	0.11 [s]
both	3626.71	1942.03	6721.38 [s]

5) Liczba studentów: 500
Liczba akademików: 5
Liczba wydziałów: 10
max_iterations: 1000
tabu_list_size: 100





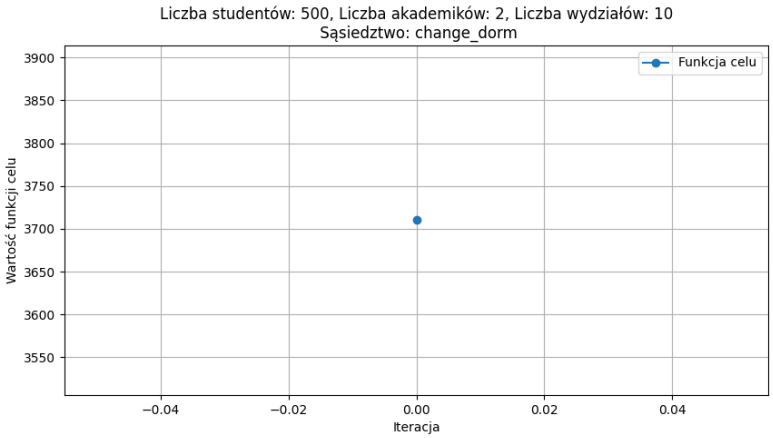


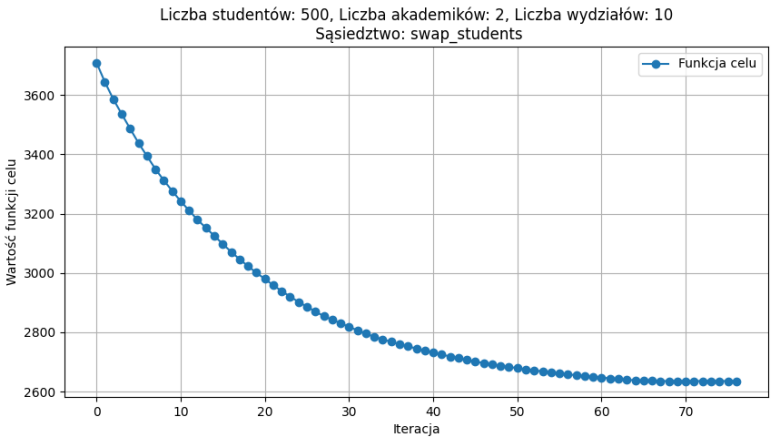


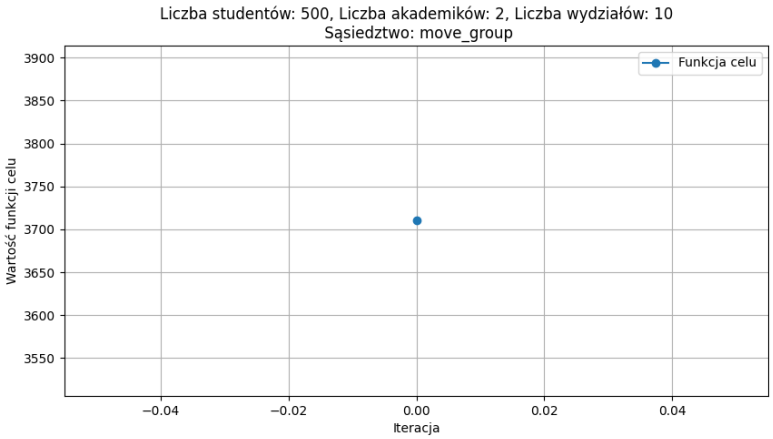
Sąsiedztwo	Wartość początkowa funkcji celu	Wartość końcowa funkcji celu	Czas obliczeń
change dorm	3727.42	2096.52	100.76 [s]
swap students	3727.42	2319.37	3767.17 [s]
move group	3727.42	3544.08	0.02 [s]

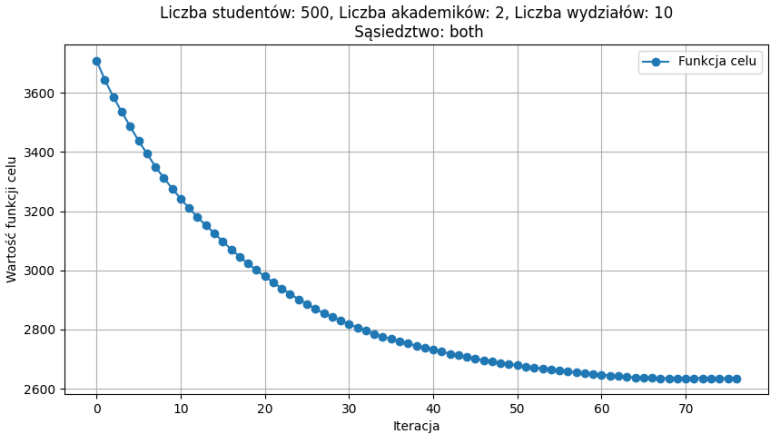
both	3727.42	2088.96	4650.52 [s]

6) Liczba studentów: 500
Liczba akademików: 2
Liczba wydziałów: 10
max_iterations: 1000
tabu_list_size: 100









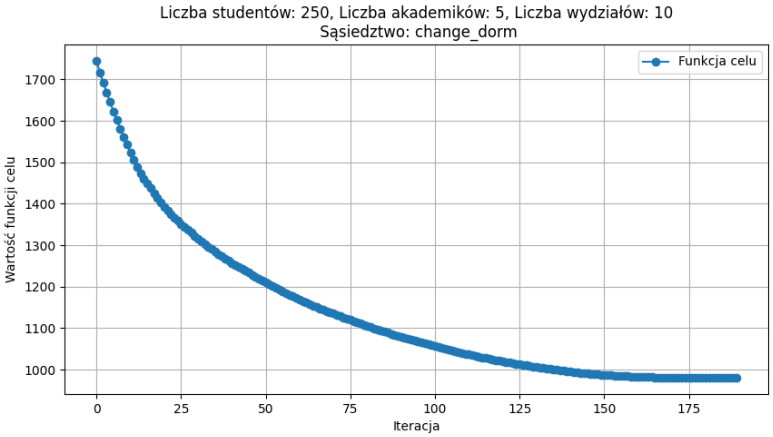
Sąsiedztwo	Wartość początkowa funkcji celu	Wartość końcowa funkcji celu	Czas obliczeń
change dorm	3710.01	3710.01	0.01 [s]
swap students	3710.01	2635.44	182.65 [s]
move group	3710.01	3710.01	0.01 [s]
both	3710.01	2635.44	183.20 [s]

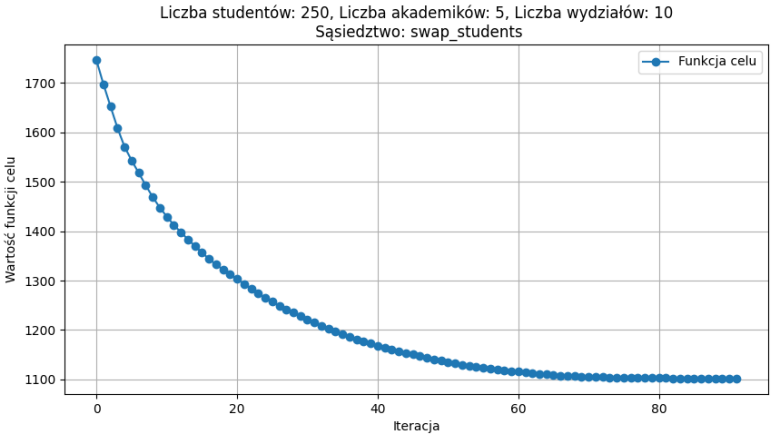
Ostatni zestaw miał na celu zbadanie wpływu argumentów *max_iterations* i *tabu_list_search* na działanie algorytm.

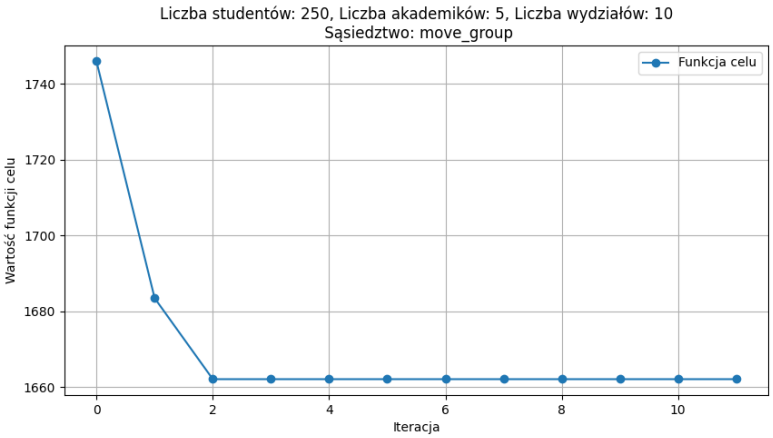
Trzeci zestaw parametrów wejściowych:

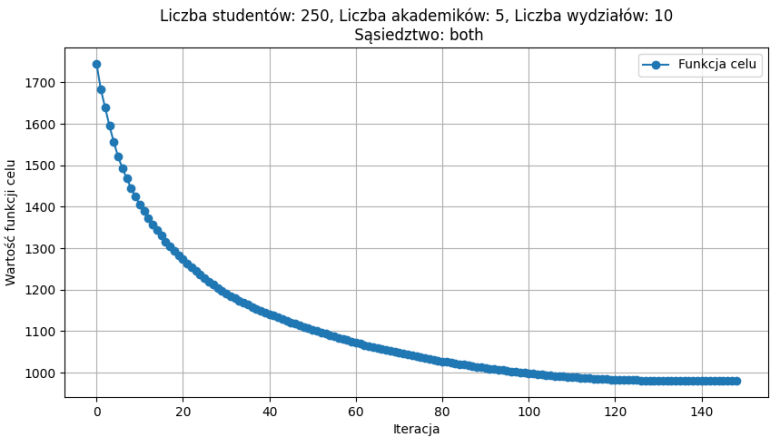
Lp.	Liczba studentów	Liczba akademików	Liczba wydziałów	*max_iterations*	*tabu_list_size*
1.	250	5	10	500	100
2.	250	5	10	500	50
3.	250	5	10	1000	50
4.	250	5	10	1000	500

1) Liczba studentów: 250
   Liczba akademików: 5
   Liczba wydziałów: 10
   max_iterations: 500
   tabu_list_size: 100


Liczba studentów: 250, Liczba akademików: 5, Liczba wydziałów: 10
Sąsiedztwo: change_dorm
Funkcja celu
Wartość funkcji celu
Iteracja

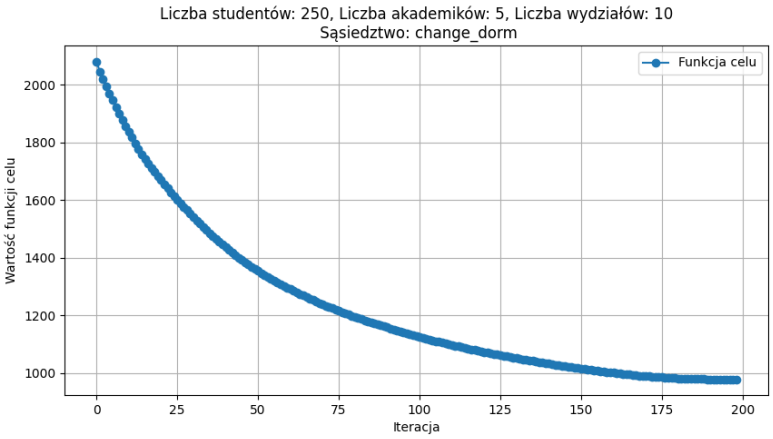

Liczba studentów: 250, Liczba akademików: 5, Liczba wydziałów: 10
Sąsiedztwo: swap_students
Funkcja celu
Wartość funkcji celu
Iteracja

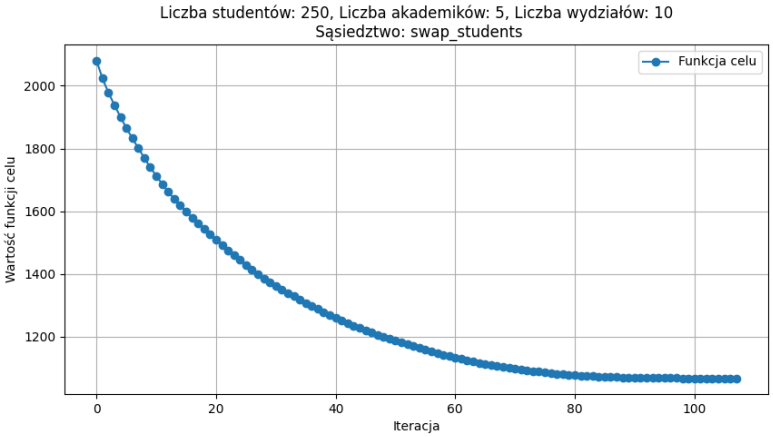

Liczba studentów: 250, Liczba akademików: 5, Liczba wydziałów: 10
Sąsiedztwo: move_group
Funkcja celu
Wartość funkcji celu
Iteracja

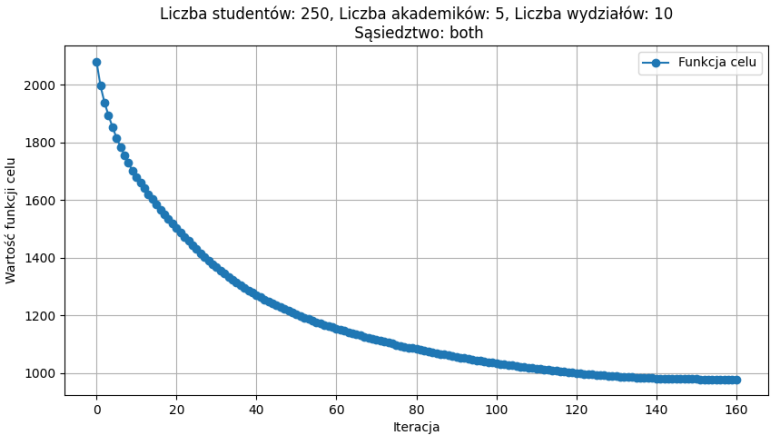


Sąsiedztwo	Wartość początkowa funkcji celu	Wartość końcowa funkcji celu	Czas obliczeń
change dorm	1745.92	979.50	15.22 [s]
swap students	1745.92	1745.92	197.35 [s]
move group	1745.92	1662.10	0.01 [s]
both	1745.92	979.50	307.34 [s]

2) Liczba studentów: 250
Liczba akademików: 5
Liczba wydziałów: 10
max_iterations: 500
tabu_list_size: 50

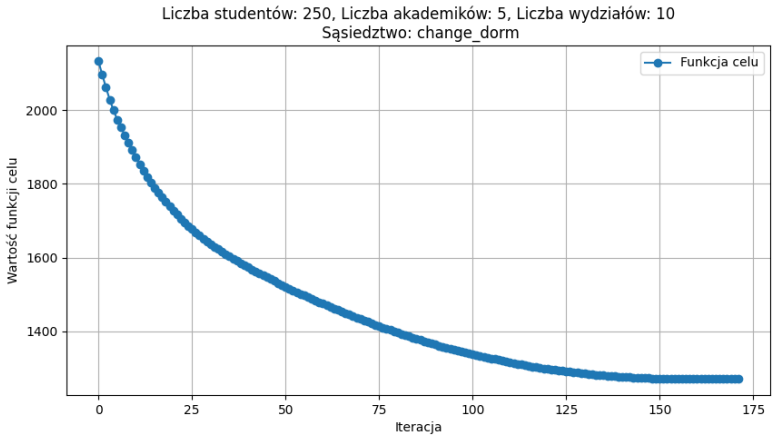


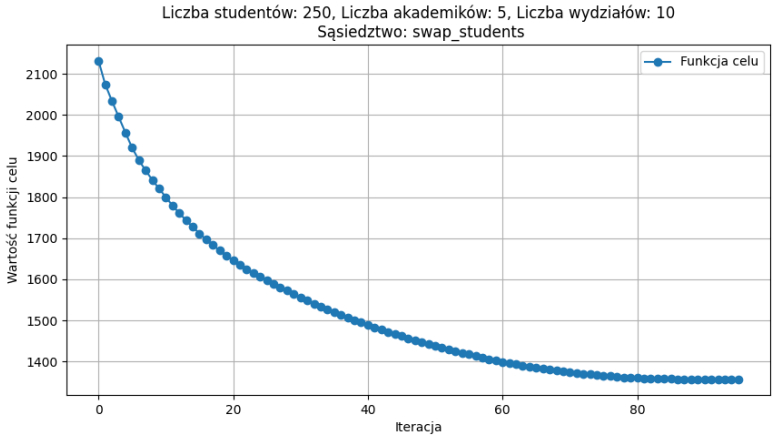


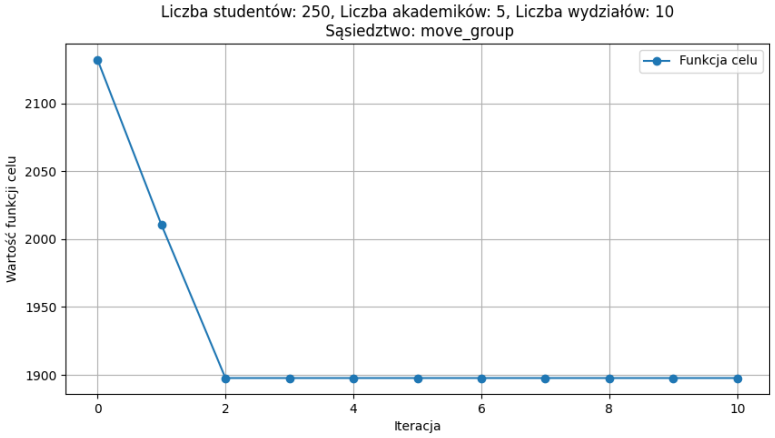


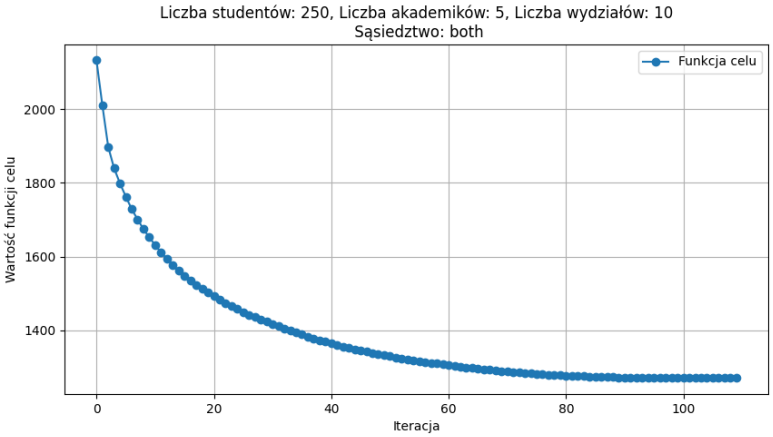
Sąsiedztwo	Wartość początkowa funkcji celu	Wartość końcowa funkcji celu	Czas obliczeń
change dorm	2080.74	979.52	16.01 [s]
swap students	2080.74	1067.77	233.38 [s]
move group	2080.74	1996.158	0.01 [s]
both	2080.74	979.52	346.98 [s]

3) Liczba studentów: 250
Liczba akademików: 5
Liczba wydziałów: 10
max_iterations: 1000
tabu_list_size: 50



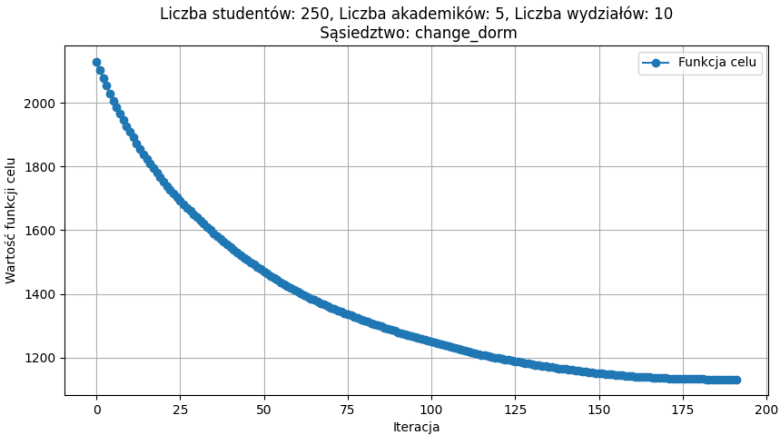


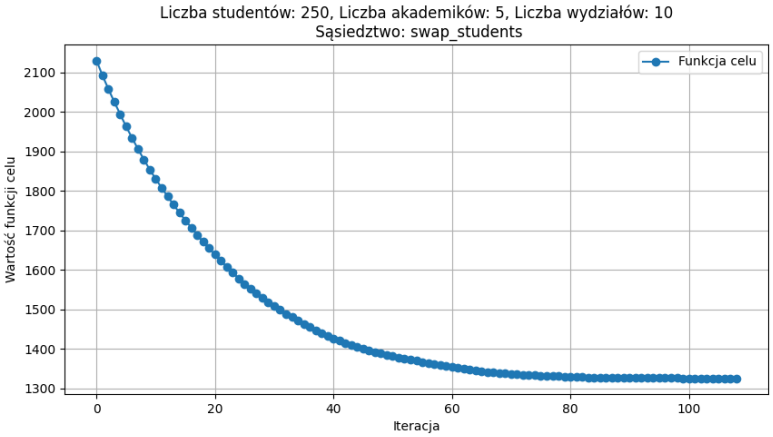


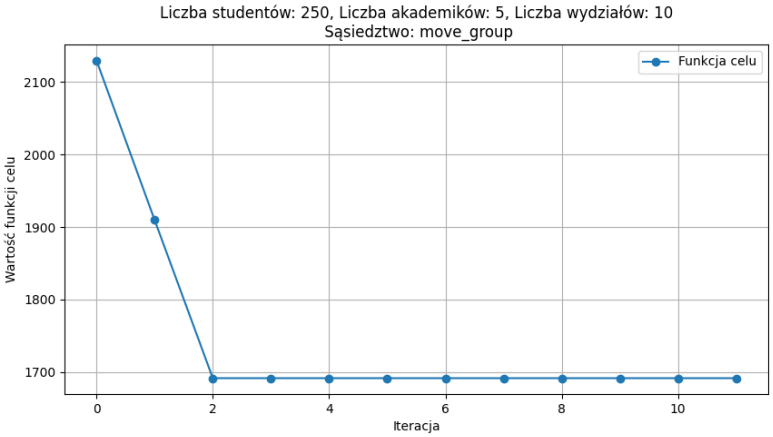


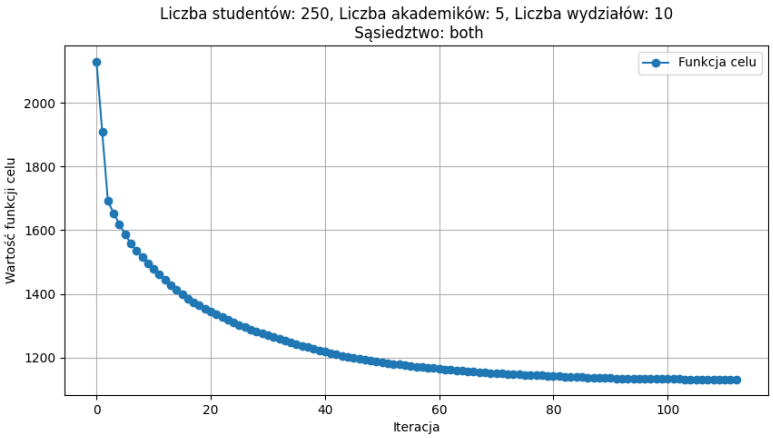
Sąsiedztwo	Wartość początkowa funkcji celu	Wartość końcowa funkcji celu	Czas obliczeń
change dorm	2132.24	1270.36	13.82 [s]
swap students	2132.24	1357.64	203.31 [s]
move group	2132.24	1897.68	0.01 [s]
both	2132.24	1270.36	219.37 [s]

4) Liczba studentów: 250
   Liczba akademików: 5
   Liczba wydziałów: 10
   max_iterations: 1000
   tabu_list_size: 500









Sąsiedztwo	Wartość początkowa funkcji celu	Wartość końcowa funkcji celu	Czas obliczeń
change dorm	2130.09	1132.49	15.74 [s]
swap students	2130.09	1325.52	231.99 [s]
move group	2130.09	1691.24	0.01 [s]
both	2130.09	1132.49	214.79 [s]

Z powyższych testów wynika, że algorytm znajduje najlepsze rozwiązania w przypadku zastosowania sąsiedztwa *change dorm*. W przypadku tego sąsiedztwa algorytm jest też optymalny pod względem czasu trwania obliczeń. Zdarzały się jednak zestawy danych, dla których algorytm nie potrafił wyznaczyć żadnego sąsiedztwa i zatrzymywał się przy pierwszej iteracji.

Zastosowanie sąsiedztwa *swap students* znacząco spowalnia program, przez co staje się on nieoptymalny dla większych zestawów danych. W większości przypadków znajduje również gorsze rozwiązanie.

Największy problem algorytm miał przy użyciu sąsiedztwa *move group*. Było to spowodowane sposobem definiowania tego sąsiedztwa. Przenoszona jest grupa studentów znajdująca się w konkretnym akademiku. Nie zawsze możliwym jest znalezienie miejsca dla grupy, przy zachowaniu ograniczeń pojemnościowych akademików.

Zastosowanie wszystkich sąsiedztw naraz pokazuje, że najlepsze rozwiązanie daje sąsiedztwo *change dorm* jeśli algorytm był w stanie je wygenerować. Jeżeli nie – najlepsze rozwiązanie znaleziono przez *swap students*. Użycie *move group* dawało przewagę w początkowych iteracjach – różnice między najlepszymi znalezionymi rozwiązaniami w kolejnych iteracjach były większe.

Zmienianie wartości parametrów *max_iterations* oraz *tabu_list_size* nie wpłynęło znacząco na wyniki obliczeń.

3. Podsumowanie

3.1 Wnioski

Algorytm optymalnego przydzielania akademików, oparty na metodzie Tabu Search, wykazuje wysoką skuteczność w znajdowaniu rozwiązań spełniających wszystkie kryteria przydziału studentów do akademików. Uwzględnienie priorytetów studentów, odległości między akademikami, a wydziałami oraz stopnia niepełnosprawności pozwala na tworzenie realistycznych modeli odpowiadających rzeczywistym potrzebom. Algorytm pozwala również na elastyczne sterowanie wagami poszczególnych kryteriów (np. priorytetów studentów), co umożliwia jego adaptację do różnych sytuacji.

3.2 Stwierdzone problemy

Wydłużony czas obliczeń dla dużych instancji problemu: Algorytm Tabu Search, mimo swojej efektywności w przestrzeni poszukiwań, wymaga znacznych zasobów obliczeniowych w przypadku rzeczywistych danych. Dla problemów obejmujących

tysiące studentów i dziesiątki akademików czas przetwarzania może być zbyt długi dla praktycznych zastosowań.

Parametry: Wynik algorytmu zależy od doboru parametrów, takich jak wielkość listy tabu, maksymalna liczba iteracji czy waga kryteriów w funkcji celu. Nieoptymalny dobór tych parametrów może prowadzić do gorszych wyników lub wydłużenia czasu obliczeń.

Ograniczona elastyczność modelu: Model nie uwzględnia jeszcze wszystkich możliwych ograniczeń rzeczywistych, takich jak współdzielenie pokoi przez studentów, specyficzne potrzeby studentów z niepełnosprawnościami, czy dynamiczna zmiana preferencji w trakcie semestru.

3.3 Kierunki dalszego rozwoju

Optymalizacja algorytmu: Wdrożenie zaawansowanych technik przyspieszających, takich jak równoległe obliczenia czy heurystyki inicjalizacyjne, mogłoby skrócić czas obliczeń dla dużych instancji problemu.

Hybrydowe podejścia: Połączenie Tabu Search z innymi metodami optymalizacyjnymi, takimi jak algorytmy genetyczne lub symulowane wyżarzanie, może poprawić jakość rozwiązań i szybkość algorytmu.

Rozbudowa modelu: Dodanie nowych funkcjonalności, takich jak uwzględnienie współdzielenia pokoi, bardziej szczegółowe kryteria lokalizacji czy zmienne preferencje studentów w czasie, pozwoliłoby na stworzenie bardziej wszechstronnego narzędzia.

Testy na rzeczywistych danych: Przeprowadzenie testów na rzeczywistych danych z uczelni mogłoby pomóc w lepszym dostrojeniu parametrów modelu oraz identyfikacji dodatkowych potrzeb użytkowników.