# Aplikacja do przydzielania akademików studentom - sprawozdanie końcowe

*Adam Palka, Bartosz Pajor, Kamil Nowobilski*

Spis treści

Aplikacja do przydzielania akademików studentom - sprawozdanie końcowe 1

1. Podział pracy 1

2. Sprawozdanie 2

2.1 Model zagadnienia - krótki opis 2

2.2 Model zagadnienia - model matematyczny 3

2.2.1 Struktury danych 3

2.2.2 Postać rozwiązania 3

2.2.3 Wejście 4

2.2.4 Funkcja celu 4

2.2.5 Ograniczenia 5

2.3 Algorytm 5

2.4 Aplikacja 11

2.5 Testy 13

3. Podsumowanie 39

3.1 Wnioski 39

3.2 Stwierdzone problemy 39

3.3 Kierunki dalszego rozwoju 40

## 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  [30%] | liczenie odległości, rozwiązanie początkowe, priorytet ze wzgl. na niepełnosprawność  [30%] | funkcja celu, opracowanie GUI  [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 |

## 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:

– liczba studentów

– rok studiów studenta

– określenie niepełnosprawności studenta

– tablica priorytetu akademików studenta

– położenie wydziału, na którym studiuje student

Zmienne dotyczące akademików:

– liczba akademików

– pojemność j-tego akademika

– położenie akademika

– odległość z wydziału studenta do akademika

#### 2.2.2 Postać rozwiązania

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

– wektor akademików

– akademik – tego studenta

#### 2.2.3 Wejście

Każdy student podaje listę priorytetu akademików w których chce mieszkać:

#### 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.

gdzie:

– priorytet wyznaczany na podstawie stopnia niepełnosprawności

– pozycja akademika na liście

– współczynnik regulujący wpływ preferencji studenta.

#### 2.2.5 Ograniczenia

Każdy akademik ma ograniczoną pojemność :

gdzie to funkcja zwracająca , gdy . W przeciwnym razie zwraca .

Liczba miejsc w akademikach jest większa niż liczba wszystkich studentów:

### 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

1. *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

1. *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

1. *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)

1. *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.

1. **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

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

|  |  |  |  |
| --- | --- | --- | --- |
| 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] |

1. Liczba studentów: 250

Liczba akademików: 5

Liczba wydziałów: 5

*max\_iterations:* 1000

*tabu\_list\_size:* 100

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

|  |  |  |  |
| --- | --- | --- | --- |
| 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] |

1. Liczba studentów: 250

Liczba akademików: 2

Liczba wydziałów: 5

*max\_iterations:* 1000

*tabu\_list\_size:* 100

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, zrzut ekranu

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, zrzut ekranu, numer

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

|  |  |  |  |
| --- | --- | --- | --- |
| 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] |

1. Liczba studentów: 250

Liczba akademików: 10

Liczba wydziałów: 10

*max\_iterations:* 1000

*tabu\_list\_size:* 100

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

|  |  |  |  |
| --- | --- | --- | --- |
| 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] |

1. Liczba studentów: 250

Liczba akademików: 5

Liczba wydziałów: 10

*max\_iterations:* 1000

*tabu\_list\_size:* 100

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, paragon, linia, Wykres

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

|  |  |  |  |
| --- | --- | --- | --- |
| 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] |

1. Liczba studentów: 250

Liczba akademików: 2

Liczba wydziałów: 10

*max\_iterations:* 1000

*tabu\_list\_size:* 100

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, zrzut ekranu

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, zrzut ekranu, numer

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

|  |  |  |  |
| --- | --- | --- | --- |
| 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

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, zrzut ekranu

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

|  |  |  |  |
| --- | --- | --- | --- |
| 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] |

1. Liczba studentów: 500

Liczba akademików: 5

Liczba wydziałów: 5

*max\_iterations:* 1000

*tabu\_list\_size:* 100

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, zrzut ekranu, numer

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

|  |  |  |  |
| --- | --- | --- | --- |
| 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] |

1. Liczba studentów: 500

Liczba akademików: 2

Liczba wydziałów: 5

*max\_iterations:* 1000

*tabu\_list\_size:* 100

Obraz zawierający tekst, linia, diagram, numer

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, zrzut ekranu

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, diagram, zrzut ekranu

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, zrzut ekranu

Opis wygenerowany automatycznie

|  |  |  |  |
| --- | --- | --- | --- |
| 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] |

1. Liczba studentów: 500

Liczba akademików: 10

Liczba wydziałów: 10

*max\_iterations:* 1000

*tabu\_list\_size:* 100

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, numer

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

|  |  |  |  |
| --- | --- | --- | --- |
| 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] |

1. Liczba studentów: 500

Liczba akademików: 5

Liczba wydziałów: 10

*max\_iterations:* 1000

*tabu\_list\_size:* 100

Obraz zawierający tekst, linia, Wykres, zrzut ekranu

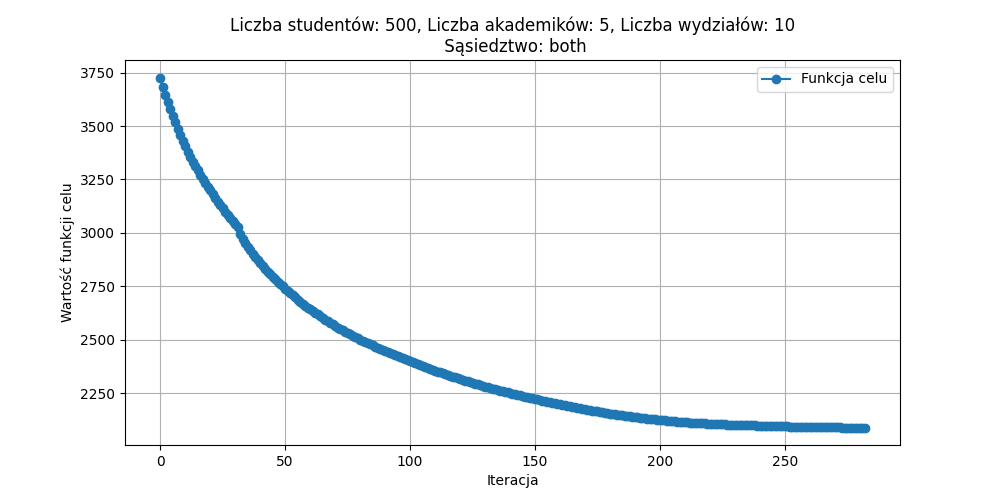
Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, numer

Opis wygenerowany automatycznie



|  |  |  |  |
| --- | --- | --- | --- |
| 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] |

1. Liczba studentów: 500

Liczba akademików: 2

Liczba wydziałów: 10

*max\_iterations:* 1000

*tabu\_list\_size:* 100

**Obraz zawierający tekst, linia, zrzut ekranu, numer

Opis wygenerowany automatycznie**

**Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie**

**Obraz zawierający tekst, zrzut ekranu, linia, numer

Opis wygenerowany automatycznie**

**Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie**

|  |  |  |  |
| --- | --- | --- | --- |
| 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

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

|  |  |  |  |
| --- | --- | --- | --- |
| 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] |

1. Liczba studentów: 250

Liczba akademików: 5

Liczba wydziałów: 10

*max\_iterations:* 500

*tabu\_list\_size:* 50

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

|  |  |  |  |
| --- | --- | --- | --- |
| 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] |

1. Liczba studentów: 250

Liczba akademików: 5

Liczba wydziałów: 10

*max\_iterations:* 1000

*tabu\_list\_size:* 50

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

|  |  |  |  |
| --- | --- | --- | --- |
| 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] |

1. Liczba studentów: 250

Liczba akademików: 5

Liczba wydziałów: 10

*max\_iterations:* 1000

*tabu\_list\_size:* 500

**Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie**

**Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie**

**Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie**

**Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie**

|  |  |  |  |
| --- | --- | --- | --- |
| 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.

## 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.