Badania Operacyjne
Optymalizacja ruchu komunikacji miejskiej

Tomasz Boroń

Grzegorz Legęza

Ryszard Pręcikowski

Paweł Lewkowicz

21 czerwca 2022

Spis treści

1 Cel projektu

Celem naszego projektu jest wyznaczenie optymalnego grafiku i tras dla komunikacji miejskiej z uwzględnieniem warunków ruchu miejskiego i zmiennego obłożenia na przystankach w czasie. Aby otrzymać rozwiązanie wykorzystamy dwa algorytmy stadne. Do poszukiwania optymalnej trasy autobusów użyjemy algorytm optymalizacji kolonią karaluchów. Natomiast za rozłożenie autobusów na linie będzie odpowiedzialny algorytm pszczeli.

2 Wprowadzenie

2.1 Sformułowanie problemu

Dla danych przystanków, połączeń między nimi oraz odgórnie ustalonej liczby autobusów należy wyznaczyć optymalny rozkład linii autobusowych na podstawie danych dotyczących obłożenia przystanków. Trasa jest reprezentowana przez ciąg przystanków. Wykorzystana zostanie struktura grafu nieskierowanego, zatem trasa będzie biec przez jednakowe przystanki w obu kierunkach. Miasto zawiera specjalne przystanki, które są punktami przesiadkowymi, na których możliwa jest zmiana linii.

Do wczytywania dostępnych połączeń między sąsiednimi przystankami, ich kosztu, a także liczby autobusów i oznaczeń punktów przesiadkowych wykorzystany będzie plik z danymi w odpowiednim formacie.

2.2 Zastosowanie

Problem ten pozwala na wyznaczenie optymalnych tras komunikacji miejskiej na podstawie znanych danych dotyczących popularności danych przystanków oraz warunków drogowych (czas przejazdu). Pozwala rozplanować ilość autobusów na daną trasę i ich czasy pojawienia się na przystankach.

2.3 Możliwe rozszerzenia problemu

- brak górnego ograniczenia na liczbę punktów przesiadkowych
- zmienna liczba dostępnych autobusów w czasie
- ograniczona liczba pasażerów w autobusie - zwiększony czas oczekiwania na kolejny, jeśli pasażer nie może wsiąść
- uwzględnienie korków, czyli różnego czasu przejazdu pomiędzy sąsiednimi przystankami w zależności od kierunku jazdy i pory dnia

2.4 Dane potrzebne do rozwiązania problemu

- przystanki - wierzchołki grafu
- bezpośrednie połączenia między przystankami - krawędzie grafu
- czas dojazdu pomiędzy bezpośrednimi przystankami - wagi krawędzi

- liczba autobusów do dyspozycji
- flagi oznaczające przystanki przesiadkowe - zakładamy, że są maksymalnie 4 punkty przesiadkowe
- grupa pasażerów z informacjami (przystanek startowy, przystanek docelowy, godzina pojawienia się na przystanku)

3 Model matematyczny problemu

3.1 Struktury danych

3.1.1 Dane

- $V = v_1, \ldots, v_n$ - przystanki, $cardV = n$
- $x_1, \ldots, x_4$ - punkty przesiadkowe (wyróżnione przystanki)
- E - zbiór krawędzi - połączeń pomiędzy przystankami
- $W((u, v))$ - czas przejazdu bezpośrednio z u do v - algorytm Dijkstry do obliczania najkrótszej ścieżki w grafie na podstawie wag krawędzi
- Θ - zbiór trójek postaci $(A, B, czas)$ - pasażer chcący się dostać z przystanku A na przystanek B, gdzie czas to godzina przyjścia na przystanek

3.1.2 Szukane

- Ω - zbiór linii, gdzie każda linia to lista przystanków - $a_1, \ldots, a_k$
- $N(linia)$ - liczba autobusów przypisana do konkretnej linii

3.2 Postać rozwiązania

Wykorzystamy dwa algorytmy, które będą działać jeden po drugim. Algorytm karalucha będzie optymalizował kształt linii autobusowych. Funkcja kosztu dla niego będzie obliczona poprzez wylosowanie kilku możliwych rozkładów autobusów i wybranie minimum po funkcji celu dla każdego z rozwiązań. Algorytm pszczeli będzie optymalizował rozmieszczenie autobusów na liniach i mierzył opisaną niżej funkcją czas przejazdu pasażerów. Dla każdego pasażera mierzmy czas oczekiwania i przejazdu do docelowego przystanku, po drodze może on się przesiadać w punktach przesiadkowych. Będziemy ten czas minimalizować.

3.3 Funkcja celu

Ω - zbiór linii autobusowych
Θ - zbiór pasażerów
$a_1, \ldots, a_k$ - rozważana linia
ST - czas na zatrzymanie autobusu na przystanku (wybierany arbitralnie, np. 1 minuta)
$T_{linia}(a_i, a_j)$ - funkcja obliczająca czas przejazdu autobusu z przystanku a_i do a_j
$T_{linia}(a_i, a_j) = \sum_{l=1}^{j-1}[W(a_l, a_{l+1}) + ST]$ dla $i < j$
$T_{linia}(a_i, a_j) = T_{linia}(a_j, a_i)$ dla $i > j$

RT - czas na zawrócenie autobusu na pętli (np. 3 minuty)
$t_{c,linia} = 2 \cdot T_{linia}(a_1, a_k) + RT$ - czas całkowity przejechania autobusu przez całą linię w obie strony.
Jeśli mamy r autobusów na linię, to muszą one ruszać co $t_{p,linia} = \frac{t_{c,linia}}{r}$

Autobusy ruszają o ustalonej godzinie T z a_0, wtedy możliwa godzina pojawienia się autobusu na przystanku a_z to ($s \in \mathbf{N_0}$):
$T + s \cdot t_{p,linia} + T_{linia}(a_1, a_z)$ - jeśli jedziemy w kierunku a_{z+1}

$T + \frac{t_{c,linia}}{2} + s \cdot t_{p,linia} + T_{linia}(a_k, a_z)$ - jeśli jedziemy w kierunku a_{z-1}

Dla każdego pasażera obliczamy minimalny czas w jakim dotrze do docelowego przystanku: $G(A, B, czas)$

Definiujemy funkcję $g(X, Y, czas)$, która określa najszybszy sposób bezpośredniego dotarcia z X do Y.
$g(X, Y, czas) = \min_{po\ l\ należącym\ do\ zbioru\ linii\ z\ X\ do\ Y}(T_l(X, Y) + czas\ oczekiwania)$

Definiujemy rodzinę ciągów $\zeta(a, b)$ jako wszystkie możliwe ciągi zaczynające się od a i kończące się na b i poza tym posiadające wyrazy należące do zbioru punktów przesiadkowych.
$\zeta(a, b) = \{(a, x_1, b), (a, b), (a, x_3, x_4, x_1, b), \dots\}$
$G(a, b, czas) = \min_{y_0,\dots,y_t \in \zeta(a,b))}(\sum_{i=0}^{t-1} h_i)$
Gdzie:
$y_0 = a, y_t = b$
$h_0 = h(y_0, y_1, czas)$
$h_{t-1} = h(y_{t-1}, y_t, czas + \sum_{i=0}^{t-2} h_i)$

Po wyprowadzeniu potrzebnych zależności możemy zapisać funkcję celu F

$F(\Omega) = \frac{1}{\#\Theta} \sum_{(A,B,czas)\in\Theta} G(A, B, czas)$

3.4 Ograniczenia

- Do każdego przystanku dojeżdża jakiś autobus: $\forall v_i$: $v_i \in V$: $\exists A \in \Omega$: $v_i \in A$
- Na każdej wyznaczonej linii kursuje przynajmniej jeden autobus: $\forall A \in \Omega$: $\exists autobus$
- Przystanki nie powtarzają się na trasie linii: $\forall A \in \Omega$: $\forall v_i, v_j \in A$ dla $i \neq j$: $v_i \neq v_j$
- Ograniczenie czasu jazdy autobusu po linii: $\forall A \in \Omega$: $\frac{t_{c,linia}}{2} < k$, gdzie k - ograniczenie czasu podróży autobusu po trasie w tę i z powrotem

4 Algorytm karalucha

4.1 Schemat algorytmu

Algorytm karalucha jest zainspirowany zachowaniem karaluchów szukających jedzenia. Każda iteracja algorytmu składa się z dwóch operacji: *chase swarming* oraz *dispersing*.

W pierwszej modelujemy podążanie za rojem, każdy karaluch podąża w kierunku najsilniejszego karalucha (najlepszego rozwiązania) w jego polu widzenia, jeśli nie widzi żadnego lepszego od siebie, to kieruje się w stronę najsilniejszego globalnego rozwiązania.

Procedura *dispersing* zapewnia różnorodność karaluchów. Polega ona na wykonaniu przez osobnika losowego ruchu.

Powyższe operacje wykonywane są w pętli do momentu spełnienia kryterium stopu, wynikiem działania algorytmu jest rozwiązanie reprezentowane przez najsilniejszego ze wszystkich karaluchów.

4.2 Adaptacja

4.2.1 Reprezentacja rozwiązania

Rozwiązanie jest reprezentowane poprzez listę wektorów. Każdy wektor odpowiada jednej linii autobusowej. Wektor składa się z numerów przystanków. Reprezentuje on ciąg przystanków na linii, gdzie pierwszy indeks wskazuje na przystanek początkowy a ostatni na końcowy.

4.2.2 Generowanie populacji początkowej

Generowanie prawidłowego rozkładu linii autobusowych składa się z kilku kroków. Przez n oznaczmy liczbę linii autobusowych. Najpierw dzielimy przystanki na n-1 podzbiorów. Do rodziny podzbiorów dokładamy zbiór złożony z przystanków przesiadkowych. Następnie z każdego zbioru tworzymy linię. Najpierw szukamy średnicy zbioru aby wyznaczyć przystanek początkowy i końcowy. Następnie rozważamy przystanek zaczynając od przystanku początkowego. Wybieramy ze zbioru przystanek, który nie został jeszcze wybrany i jest możliwie najbliżej rozważanego, i możliwe najdalej końcowego. Przystanek ten wraz z przystankami pomiędzy rozważanym i wybranym dodajemy do listy reprezentującej linię autobusową. Krok powtarzamy rozważając ostatnio wybrany aż do wybrania wszystkich przystanków. Na końcu dodajemy przystanek końcowy. Przy tworzeniu linii dbamy, aby każdy przystanek pojawił się w niej maksymalnie jeden raz.

4.2.3 Szczegóły adaptacji

Widoczne karaluchy wyznaczane są przez wyznaczenie liczby wspólnych krawędzi z każdym innym karaluchem a następnie sprawdzenie, czy ich liczba jest większa od ustalonego progu.

4.2.4 Pseudokod

Główny algorytm

1. wygeneruj populację początkową równą liczbie karaluchów (parametr), każde rozwiązanie przypisz jednemu karaluchowi
2. oblicz wartość funkcji celu dla każdego z karaluchów
3. wykonaj w pętli określoną liczbę iteracji (parametr)
   (a) wykonaj procedurę *chase swarming*
   (b) wykonaj procedurę *dispersing*
4. wyznacz najlepszego karalucha i zwróć jego rozwiązanie

Procedura *chase swarming*
Dla każdego karalucha:

1. wyznacz widoczne karaluchy
2. wyznacz najlepszego widocznego karalucha
3. jeśli rozwiązanie aktualnego jest lepsze od widocznego najlepszego to przypisz do najlepszego widocznego globalne maximum
4. wyznacz losową wspólną krawędź między aktualnym a najlepszym widocznym
5. wyznacz nową listę przystanków dla linii do której należy dana krawędź:
   (a) wyznacz przystanki pochodzące od aktualnego karalucha - przystanki do danej krawędzi
   (b) wyznacz *rozmiar kroku* (parametr) przystanków pochodzących od najlepszego widocznego karalucha - przystanki po danej krawędzi
   (c) połącz obie listy przystanków
   (d) wyznacz nową linię z listy przystanków
6. jeśli nowe rozwiązanie jest lepsze od aktualnego to zaktualizuj aktualne

Procedura *dispersing*
Dla każdego karalucha:

1. wyznacz losową linię do modyfikacji
2. wyznacz maksymalną liczbę przystanków do modyfikacji (parametr)

3. usuń z jednego końca losową liczbę przystanków $\leqslant$ liczby z poprzedniego kroku
4. zastąp usunięte nowymi, losowymi przystankami (ilość $\leqslant$ maksymalna liczba przystanków do modyfikacji)
5. wyznacz nową linię z listy przystanków
6. jeśli nowe rozwiązanie jest lepsze od aktualnego to zaktualizuj aktualne

5 Algorytm pszczeli

5.1 Schemat algorytmu

Algorytm pszczeli symuluje zachowanie roju pszczół. Najpierw każda z pszczół leci w losowe miejsce i tam sprawdza rozwiązanie. Następnie pszczoły wracają do ula i następuje przeorganizowanie pszczół. Wybierane jest k najlepszych rozwiązań (k jest obliczane poprzez przemnożenie liczby pszczół i parametru algorytmu 'update_ratio') i następnie wszystkie pszczoły rozdzielają się do tych k miejsc. W każde miejsce leci odwrotnie proporcjonalna liczba pszczół względem otrzymanego rezultatu w danym miejscu. Następnie w każdym z tych miejsc, pszczoły które tam się udały rozpraszają się w najbliższym sąsiedztwie. Pszczoła która znajdzie najlepsze rozwiązanie zostaje przypisana do tego rozwiązania. Później wszystkie niezatrudnione pszczoły lecą w losowe miejsce i tam znajduje pewne rozwiązanie. Algorytm się zapętla i znów pszczoły przylatują do ula i wybierają najlepsze rozwiązania. Algorytm trwa, aż do spełnienia kryterium stopu (np. liczby iteracji). Na koniec wybierane jest najlepsze rozwiązanie wśród wszystkich pszczół.

5.2 Adaptacja

5.2.1 Reprezentacja rozwiązania

Rozwiązanie jest reprezentowane poprzez wektor liczb całkowitych dodatnich. Wektor ma długość równą liczbie linii. Liczba na i-tej pozycji odpowiada liczbie autobusów przypisanej do i-tej linii.

5.2.2 Generowanie populacji początkowej

Generowanie populacji rozwiązań polega na wygenerowaniu wektorów liczb. Dla danego wektora ustalamy rozkład normalny i z niego losujemy próbki. Średnią ustalamy jako liczbę autobusów podzieloną przez liczbę linii. Odchylenie standardowe natomiast losujemy z przedziału $(0.5{*}\text{średnia}, 1.5{*}\text{średnia})$. Otrzymany wektor poddajemy lekkiej modyfikacji. Po pierwsze dbamy, aby każda linia otrzymała co najmniej jeden autobus. Po drugie suma liczb ma równać się liczbie dostępnych autobusów, więc aby to uzyskać z losowo wybranych linii usuwamy bądź dodajemy autobusy wedle potrzeb.

5.2.3 Przeszukiwanie sąsiedztwa

Sąsiedztwo wektora jest obliczane poprzez wykonanie zamiany przypisania autobusu do linii k-razy, gdzie k to parametr algorytmu. Zamiana polega na wylosowaniu dwóch linii autobusowych i odjęciu jednego autobusu z pierwszej z nich oraz dodanie go do drugiej. Dodatkowo dodane zostało ograniczenie na wybór pierwszej linii, aby nie uzyskać linii bez autobusów.

5.2.4 Pseudokod

1. wygeneruj populację początkową równą liczbie pszczół, każdej pszczole przypisz jedno rozwiązanie
2. oblicz wartość funkcji celu dla każdej z pszczół
3. wykonaj poniższe k razy (k - parametr algorytmu):
   (a) wybierz p najlepszych pszczół (p - parametr algorytmu)

(b) dla każdej z wybranych pszczół:
   i. oblicz, ile pszczół zostanie dodanych do tego miejsca
   ii. dla każdej dodanej pszczoły wygeneruj rozwiązanie z sąsiedztwa
   iii. oblicz funkcje celu dla rozwiązań z sąsiedztwa
   iv. pszczole z najlepszym rozwiązaniem pozostaw je, pozostałe pszczoły zwolnij

(c) wygeneruj populację początkową równą liczbie wolnych pszczół, każdej wolnej pszczole przypisz jedno rozwiązanie

(d) oblicz wartość funkcji celu dla każdej z pszczół

4. wybierz pszczołę z najlepszym rozwiązaniem i je zwróć

6 Aplikacja

6.1 Technologie

Aplikacja została w całości napisana w języku Python. Dane do programu wczytujemy z pliku .json. Algorytmy zostały zaimplementowane z wykorzystaniem modułu Numpy. Wykresy powstały za pomocą narzędzi z biblioteki Matplotlib. Wizualizacja grafów została napisana z wykorzystaniem modułu Graphviz.

6.2 Uruchomienie

Aby uruchomić aplikację należy w katalogu głównym aplikacji wywołać poniższe komendy:

```
virtualenv .venv
source .venv/bin/activate
python3 -m pip install -r requirements.txt
```

Następnie możliwe są trzy sposoby uruchomienia aplikacji. Po pierwsze można wygenerować przykładowe dane wejściowe dla naszego problemu. Aby ustawić parametry nowego grafu należy zedytować sekcję 'generate' w piku konfiguracyjnym. Przykładowe wywołanie to:

```
python3 ./src/main.py <filepath> -g
```

Aby uruchomić algorytmy należy najpierw ustawić odpowiednie parametry w pliku konfiguracyjnym, a następnie wywołać komendę:

```
python3 ./src/main.py <filepath> -s
```

Dodatkowo istnieje możliwość testowania parametrów. W tym celu należy odpowiednio zapisać kryterium w pliku konfiguracyjnym w sekcji 'criterias' i następnie wywołać komendę:

```
python3 ./src/main.py <filepath> -t <name of criteria>
```

6.3 Konfiguracja aplikacji

Poniżej prezentujemy plik konfiguracyjny naszej aplikacji wraz z opisem poszczególnych parametrów.

Listing 1: Plik konfiguracyjny: src/config.py

```
config = {
    'num_lines': 4,
    'num_buses': 20,
    'num_tests': 4,
    'generate': {
        'vertices': 20,
        'min_edges': 30,
        'num_passengers': 80
    },
```

```
    'cockroach': {
        'num_cockroaches': 10,
        'min_common': 8,
        'step_size': 2,
        'dispersing_update_ratio': .5,
        'n_iterations': 10,
        'num_to_test': 5
    },

    'bees': {
        'num_bees': 10,
        'num_transition': 2,
        'update_ratio': .4,
        'n_iterations': 10
    }
}

criterias = {
    '1': {
        'cockroach': {
            'num_cockroaches': [5, 10, 15, 20],
        },
    },
    '2': {
        'cockroach': {
            'min_common': [1, 4, 7, 10],
        },
    },
    '3': {
        'cockroach': {
            'step_size': [1, 2, 4, 8],
        },
    },
    '4': {
        'cockroach': {
            'dispersing_update_ratio': [0.2, 0.3, 0.4, 0.5],
        },
    },
    '5': {
        'bees': {
            'num_bees': [5, 10, 15, 20],
        },
    },
    '6': {
        'bees': {
            'num_transition': [1, 2, 3, 4],
        },
    },
    '7': {
        'bees': {
            'update_ratio': [.3, .4, .5, .6],
        },
    }
}
```

Ustawienia globalne	
num_lines	liczba lini autobusowych
num_buses	liczba dostępnych autobusów
num_tests	liczba testów dla opcji testowania parametrów
Ustawienia generatora	
vertices	liczba przystanków
min_edges	minimalna liczba bezpośrednich połączeń pomiędzy przystankami
num_passengers	liczba pasażerów
Algorytm karalucha	
num_cockroaches	liczba karaluchów
min_common	minimalna liczba wspólnych krawędzi
step_size	rozmiar pojedynczego kroku
dispersing_update_ratio	maksymalny procent krawędzi do modyfikacji
n_iterations	liczba iteracji
num_to_test	liczba rozkładów busów do metryki
Algorytm pszczeli	
num_bees	liczba pszczół
num_transition	liczba tranzycji przy przeszukiwaniu sąsiedztwa
update_ratio	procent pszczół, które mają najlepsze rozwiązania
n_iterations	liczba iteracji

Tabela 1: Słownik oznaczeń pliku config.py

7 Testy

Do testów wykorzystaliśmy kryteria z pliku konfiguracyjnego zamieszczonego wyżej. Pozostałe parametry miały wartości domyślne jak w pliku wyżej.

7.1 Tabele

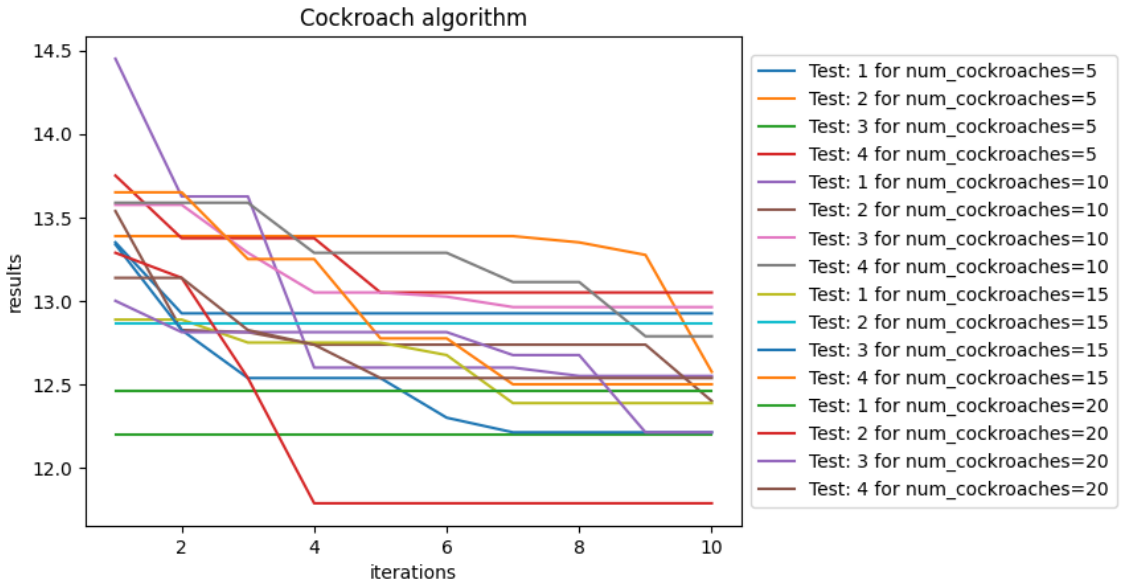
Wpływ liczności populacji			
Liczba karaluchów	Najgorsza wartość	Najlepsza wartość	Najlepsze znalezione rozwiązanie
5	12.71	11.85	11.49
10	12.49	11.49	
15	12.46	11.75	
20	12.98	11.65	
Wpływ liczby wspólnych krawędzi			
Wspólne krawędzie	Najgorsza wartość	Najlepsza wartość	Najlepsze znalezione rozwiązanie
1	12.50	11.86	11.79
4	12.33	11.83	
7	12.63	11.79	
10	12.66	12.08	
Wpływ rozmiaru kroku			
Rozmiar	Najgorsza wartość	Najlepsza wartość	Najlepsze znalezione rozwiązanie
1	13.15	11.57	11.10
2	12.56	11.10	
4	12.13	11.81	
8	12.53	12.00	
Wpływ współczynnika aktualizacji w procedurze dispersing			
Współczynnik aktualizacji	Najgorsza wartość	Najlepsza wartość	Najlepsze znalezione rozwiązanie
0.2	12,24	11,69	11,23
0.3	12,08	11,23	
0.4	12,81	11,39	
0.5	12,56	11,41	

Tabela 2: Testy dla algorytmu karalucha

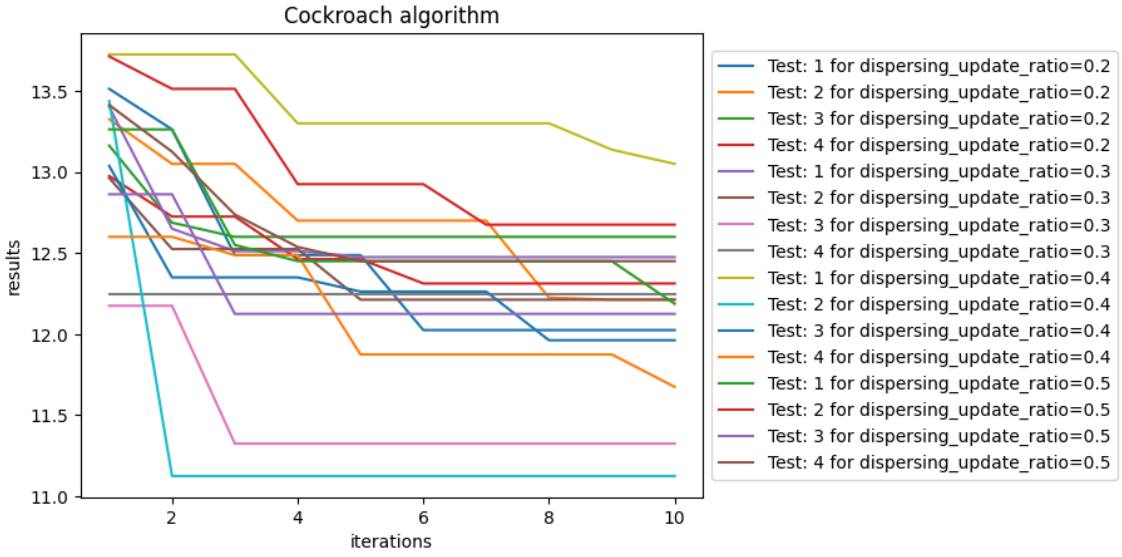
Wpływ liczności populacji			
Populacja rozwiązań	Najgorsza wartość	Najlepsza wartość	Najlepsze znalezione rozwiązanie
5	12,68	12,03	11,34
10	12,86	11,34	
15	12,18	12,00	
20	12,74	11,34	
Wpływ liczby tranzycji			
Liczba tranzycji	Najgorsza wartość	Najlepsza wartość	Najlepsze znalezione rozwiązanie
1	11.98	11.69	11.35
2	12.49	11.35	
3	12.55	11.76	
4	12.93	11.54	
Wpływ współczynnika wyboru			
Wartość	Najgorsza wartość	Najlepsza wartość	Najlepsze znalezione rozwiązanie
0.3	12.69	11.71	11.21
0.4	12.48	11.74	
0.5	12.38	11.21	
0.6	12.73	12.21	

Tabela 3: Testy dla algorytmu pszczelego

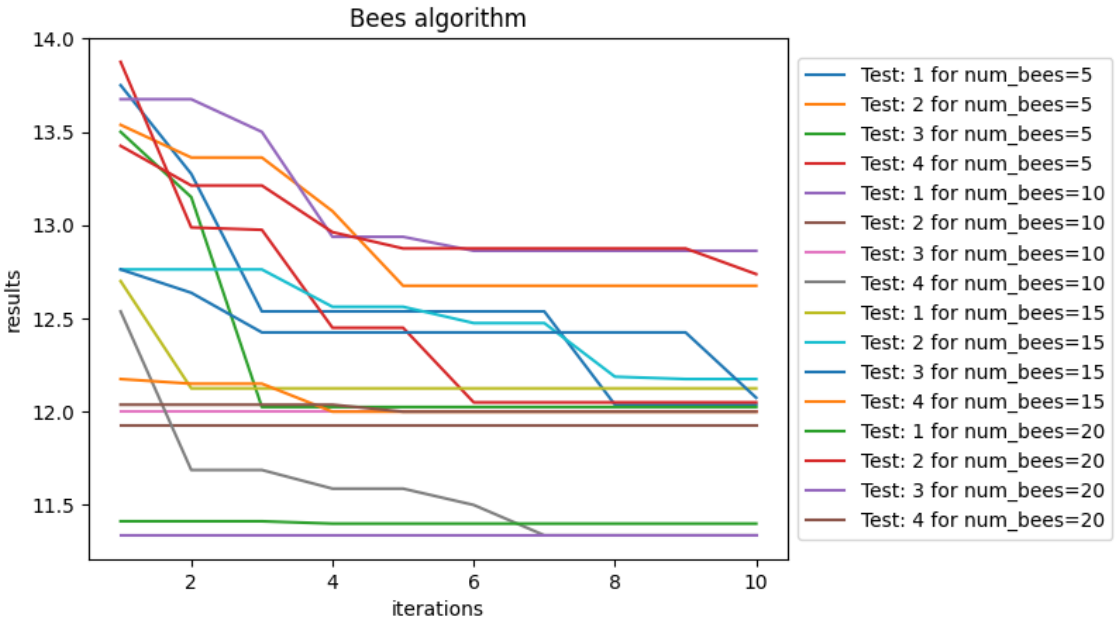
7.2 Wykresy



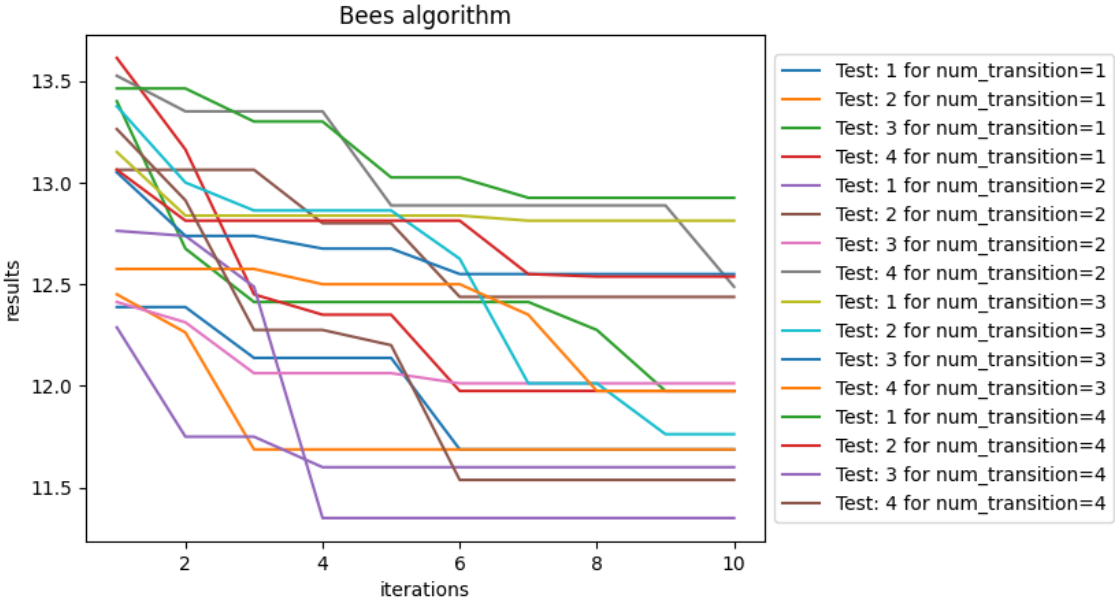
Rysunek 1: Test wpływu liczności populacji w algorytmie karalucha



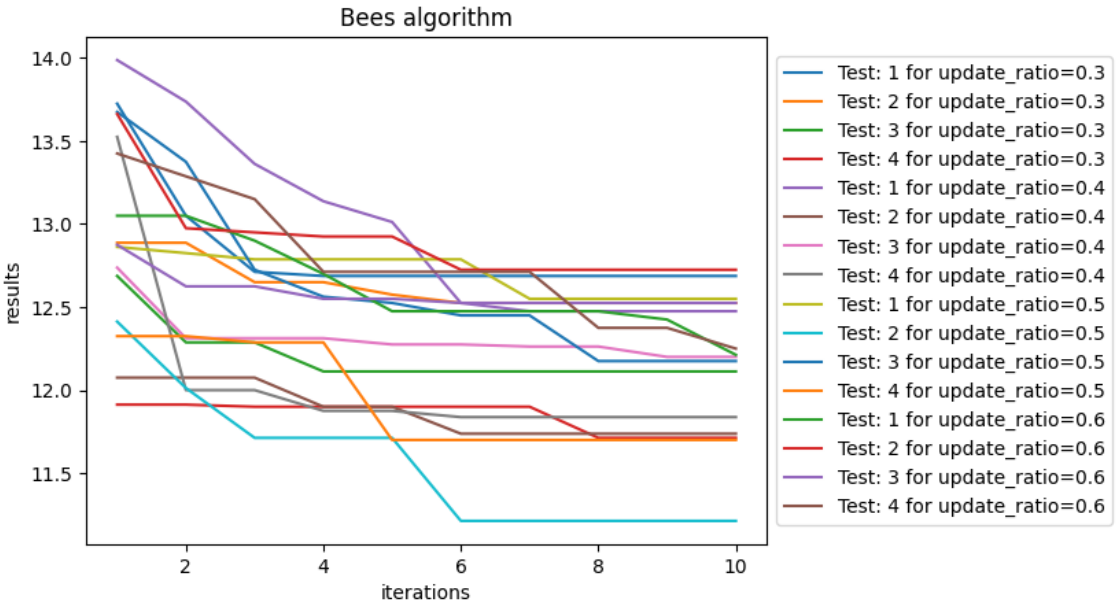
Rysunek 2: Test wpływu współczynnika aktualizacji w procedurze dispersing



Rysunek 3: Test wpływu liczności populacji w algorytmie pszczelim



Rysunek 4: Test wpływu liczby tranzycji



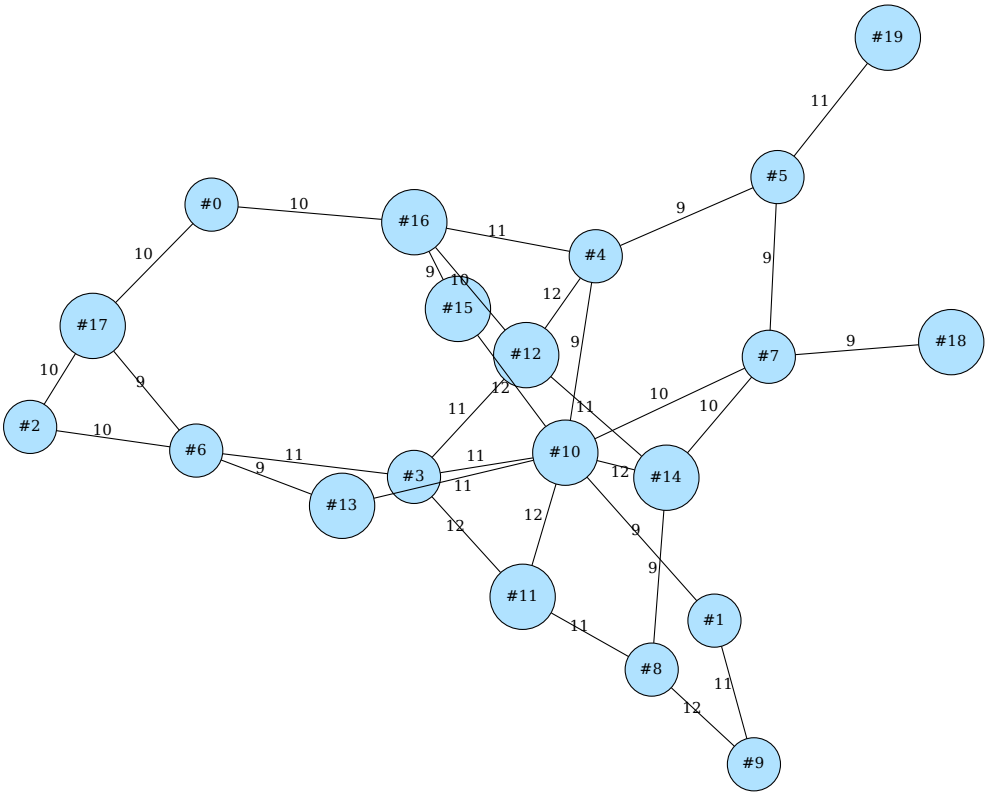
Rysunek 5: Test wpływu współczynnika wyboru

8 Przykładowe rozwiązanie problemu

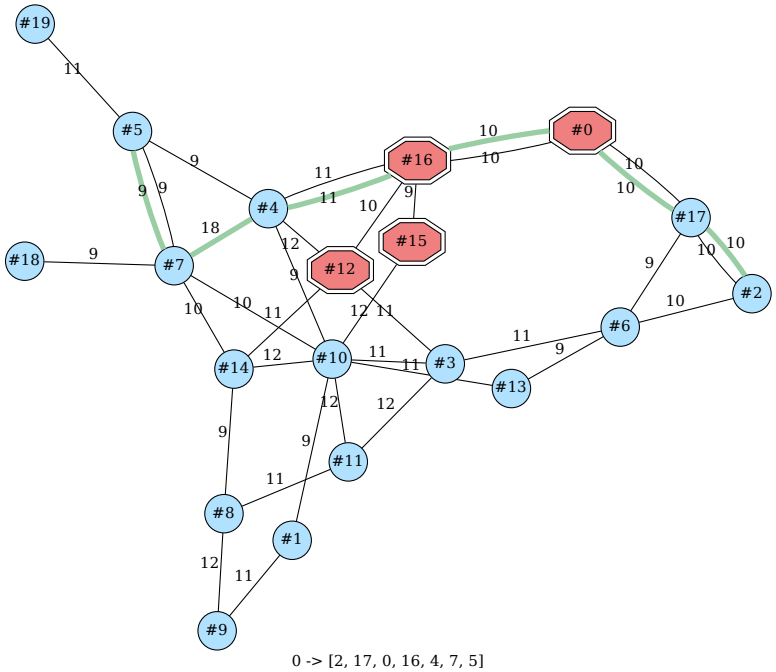
W poniżej przedstawionym problemie algorytm wystartował przy rozwiązaniu początkowym z wartością funkcji celu równą 75,59. Po 20 iteracjach algorytmu karalucha otrzymaliśmy wartość 67,34. Na koniec działania aplikacji wartość funkcji celu wyniosła 66,34 (po 20 iteracjach algorytmu pszczelego).

numer przystanku	liczba pasażerów na przystanku	liczba pasażerów zmierzających na ten przystanek	suma pasażerów
0	7	5	12
1	5	4	9
2	3	4	7
3	3	0	3
4	8	4	12
5	2	5	7
6	5	4	9
7	6	3	9
8	5	3	8
9	2	5	7
10	3	6	9
11	6	3	9
12	4	2	6
13	1	5	6
14	7	7	14
15	5	2	7
16	0	2	2
17	2	7	9
18	2	4	6
19	4	5	9

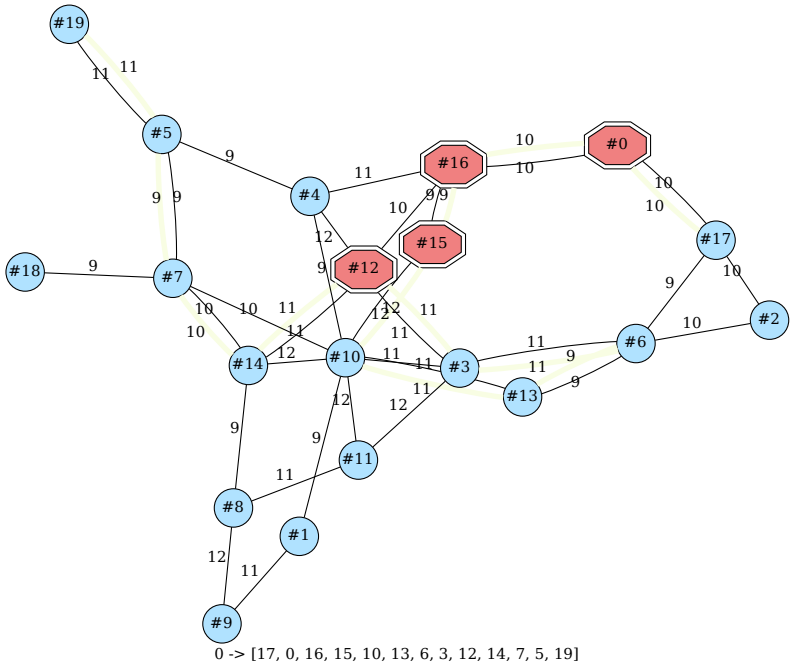
Tabela 4: Rozkład liczby pasażerów na przystankach



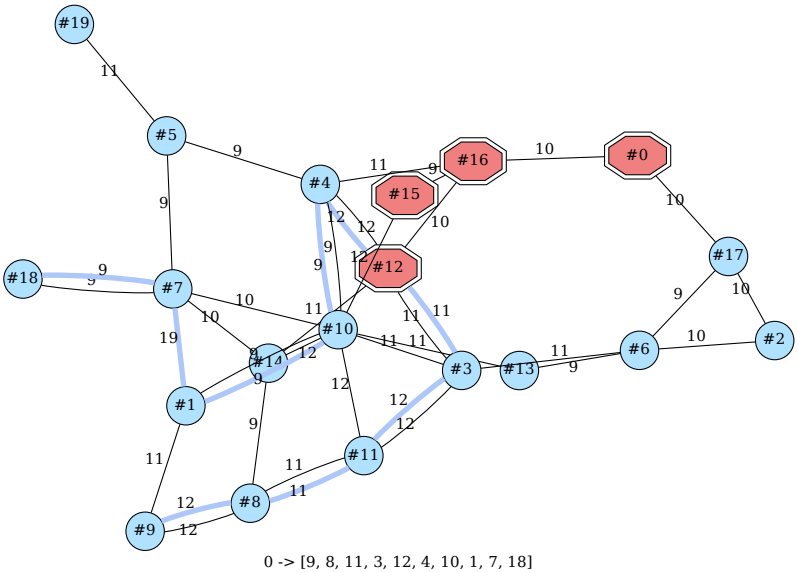
Rysunek 6: Graf dla problemu z zadania



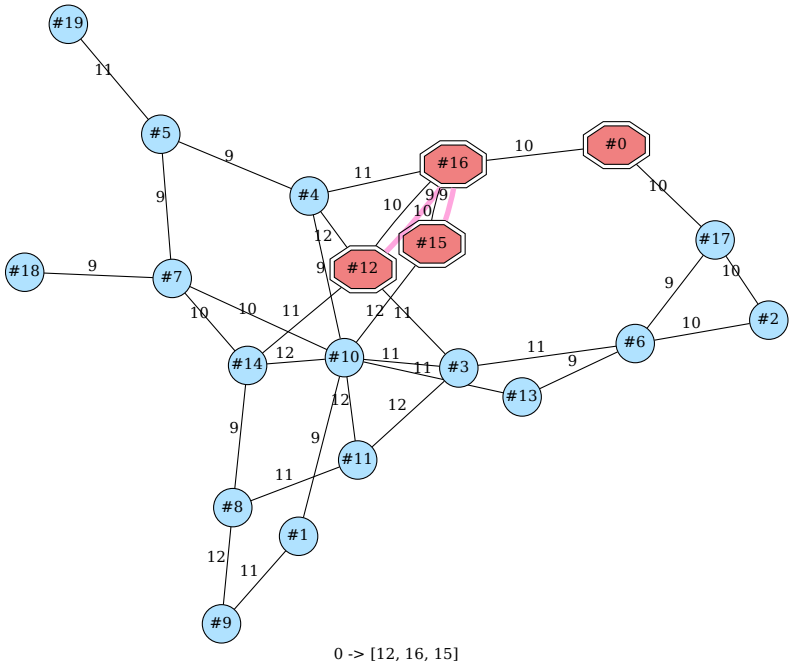
Rysunek 7: Linia 0 w rozwiązaniu, przypisano 3 autobusy



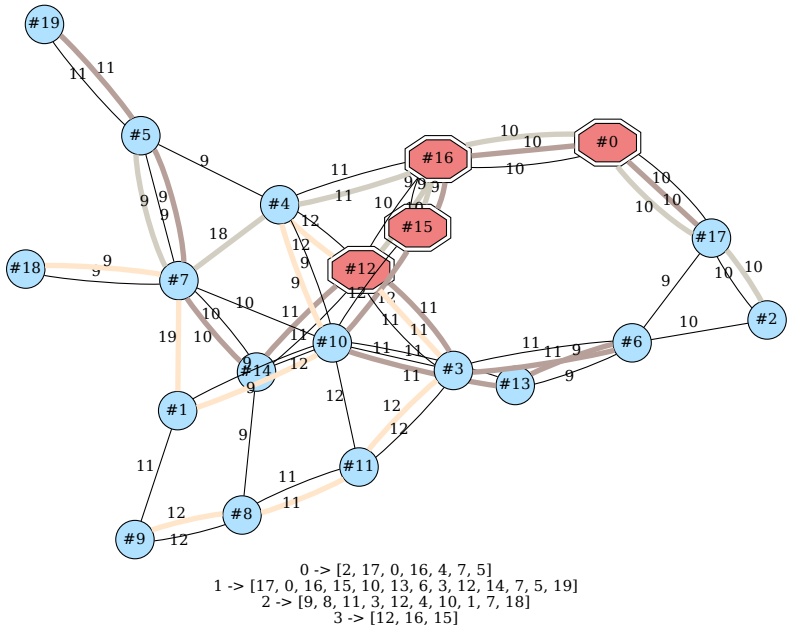
Rysunek 8: Linia 1 w rozwiązaniu, przypisano 8 autobusów



Rysunek 9: Linia 2 w rozwiązaniu, przypisano 8 autobusów



Rysunek 10: Linia 3 w rozwiązaniu, przypisano 1 autobus



Rysunek 11: Wizualizacja wszystkich lini autobusowych

9 Podsumowanie

- Na naszą aplikację duży wpływ mają elementy losowe. Dobre wygenerowanie populacji początkowej daje spore szansę na znalezienie dobrego wyniku.
- Przy zwiększonej liczbie iteracji obserwujemy znaczny wzrost czasu wykonania, ale też wartości funkcji celu powoli spadają.
- Początkowo algorytmy w naszym programie miały być zagnieżdżone. Gdybyśmy się zdecydowali na taki wybór, otrzymalibyśmy bardziej przewidywalne wyniki z powodu lepszego sposobu obliczania funkcji celu. Jednakże oznaczałoby to znaczny wzrost czasu działania algorytmu (liczba iteracji do kwadratu).
- Testując pojedyncze parametry, nie dla wszystkich da się zdefiniować zasadę poprawy rozwiązania. Wynika to głównie z faktu, że algorytm jest w pewnym stopniu randomizowany oraz tego, że przy kilku parametrach zmiana jednego niekoniecznie musi mieć znaczny wpływ na wynik.

10 Literatura

`https://www.mdpi.com/1099-4300/19/5/213/htm` - artykuł z opisem wykorzystania algorytmu karalucha do problemu planowania trasy, stanowiący bazę dla naszej implementacji