# Badanie efektywności algorytmu unikania kolizji w przestrzeni powietrznej

Wojciech Chwaciński 151924

28.04.2024

## 1.Cel eksperymentu.

Zbadać efektywność algorytmu solwera CPLEX wykorzystywanego do rozwiązywania problemu programowania matematycznego, w tym przypadku naszym problemem było wybranie odpowiedniego manewru dla każdego samolotu tak aby nie spowodowało to żadnej kolizji.

## 2. Parametry programu.

Rozmiar macierzy konfliktów :

-n -liczba samolotów

-m -liczba manewrów dostępnych dla każdego samolotu ( każdy samolot ma ich tyle samo)

Wartości wpływające na prawdopodobieństwo konfliktu:

-p -prawdopodobieństwo wystąpienia konfliktu między dwoma statkami powietrznymi poruszającymi się po pierwotnie ustalonych trasach
-w -miara „zachowania konfliktu" przy rosnącej liczbie możliwych do wykonania manewrów
-p_m -prawdopodobieństwo wystąpienia konfliktu między dwoma statkami powietrznymi dla przyjętej liczby *m* możliwych manewrów wyrażone jest wzorem:

$$p * 2^{\frac{1-m}{w}}$$

## 3. Generowanie instancji problemu.

Wykorzystując funkcje „generator_instancji()" tworzę macierz składającą się z mniejszych macierzy (m x m), które są wypełnione zerami, następnie wstawiam w jej górny trójkąt „1" z odpowiednim prawdopodobieństwem, na koniec przy użyciu funkcji „only_zeros()" usuwam konflikty z macierzy które leżą na głównej przekątnej. Dla każdej kombinacji parametrów proces ten powtarzany jest 10 razy aby otrzymać większą próbkę do badań.

```
def generator_instancji(n, m, p, w ):
    conf_matrix = []
    size =  n * m
    conf_matrix = [[0 for _ in range(size)] for _ in range(size)]
    p_m = p * pow(2, ((1 - m)/w))
    samples = []
    for i in range(size):
        for j in range(i, size):
            if i == j:
                conf_matrix[i][j] = 1
            else:
                conf_matrix[i][j] = 1 if random.random() < p_m else 0
    return conf_matrix


def only_zeros(matrix, m):
    n = len(matrix)
    for i in range(0, n, m):
        for j in range(0, n, m):
            if i == j:  # Sprawdzamy, czy jesteśmy na przekątnej
                for k in range(m):
                    for l in range(m):
                        if k != l:
                            matrix[i + k][j + l] = 0
    return matrix
```

## 4.Rozwiązywanie problemy.

Początkowo kod przechodzi przez wszystkie pliki w bieżącym folderze i wybiera te, które zaczynają się od prefiksu określonego przez zmienne n, m, p i w.
Następnie dla każdego wybranego pliku, który zawiera dane dotyczące instancji problemu, pobiera liczby z pliku, określając liczbę samolotów, liczbę manewrów i macierz związaną z dostępnymi manewrami dla każdego samolotu.
Tworzy się model programowania matematycznego.

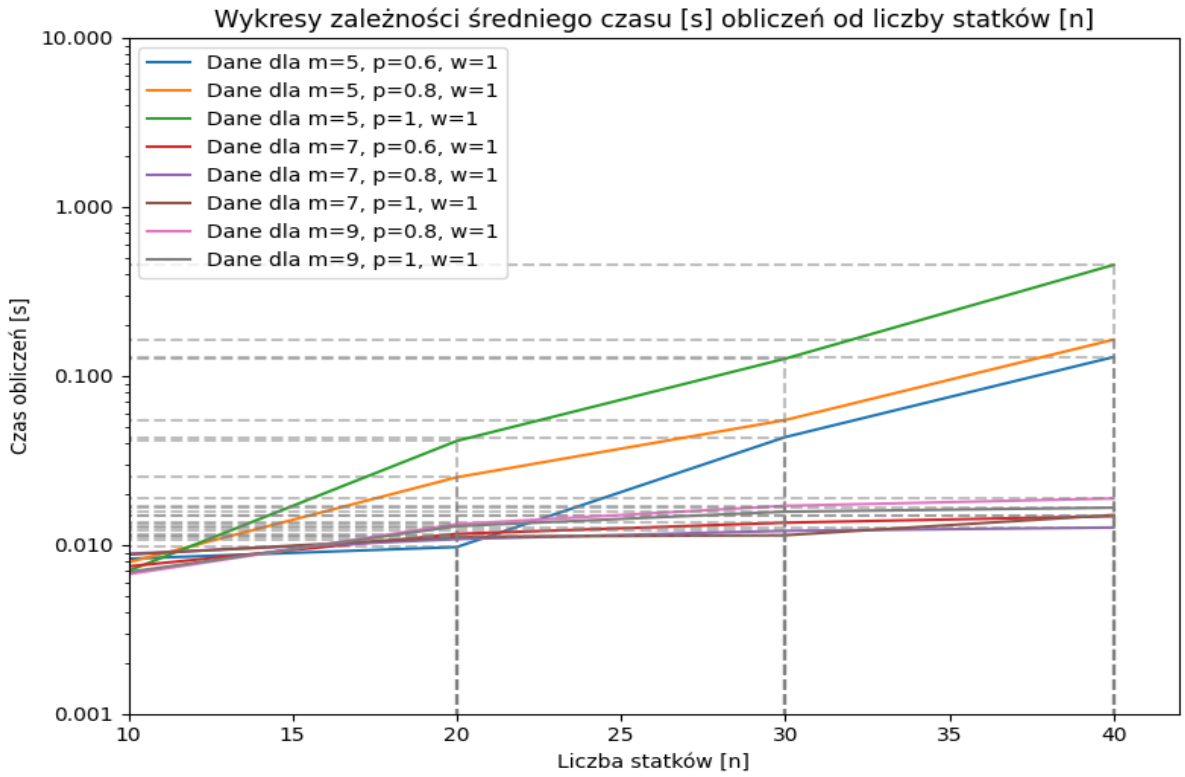
Dodawane są ograniczenia, takie jak każdy samolot wykonuje dokładnie jeden manewr oraz ograniczenia dotyczące konfliktów między manewrami różnych samolotów.
Tworzy się funkcję celu, która minimalizuje sumę czasów trwania manewrów.
Rozwiązuje się model za pomocą metody solve() i mierzy czas potrzebny na rozwiązanie problemu.
Zebrane czasy obliczeń oraz informacje o liczbie znalezionych rozwiązań dopuszczalnych zapisywane są do pliku wynikowego o nazwie zawierającej prefiks instancji problemu.

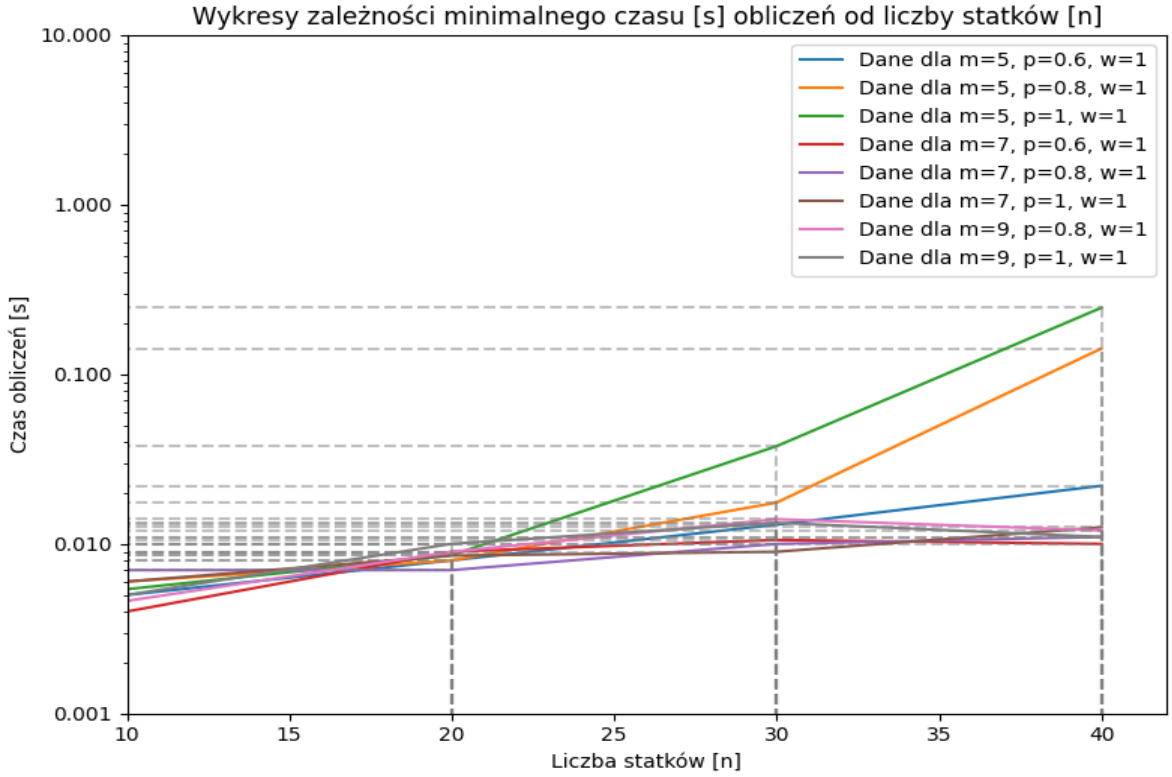
```
import re

from docplex.mp.model import Model

import time
import os
curr_folder = "."
n = [ 40]
m = [3, 5, 7, 9]
p = [0.6, 0.8, 1]
w = [1, 5, 10]
for n_x in n:
    for m_x in m:
        for p_x in p:
            for w_x in w:
                pref = f'n={n_x}_m={m_x}_p={p_x}_w={w_x}_iteracja='
                czasy = []
                ilosc_rozw_dop = []
                for file_name in os.listdir(curr_folder):
                    if file_name.startswith(pref):
                        numbers = re.findall(r'\d+', file_name)
                        file = open(file_name, 'r', encoding='Utf-8')
                        matrix = []
                        n = int(numbers[0])  # liczba samolotów
                        m = int(numbers[1])  # liczba manewrów
                        print(n, m)
                        for i in range(n*m):
                            read = file.readline()
                            num = re.findall(r'\d+', read)
                            dane = list(map(int, num))
                            matrix.append(dane)
                        model = Model()
                        model.populate_solution_pool = True
                        x = {(i, j): model.binary_var(name=f'x_{i}_{j}') for i in range(n) for j in
range(m)}

                        for i in range(n):
                            model.add_constraint(model.sum(x[i, j] for j in range(m)) == 1)
                        flaga = 0
                        for i in range(len(matrix)):
                            if i%m == 0:
                                flaga += 1
                            for j in range(flaga*m, len(matrix[i])):
                                if matrix[i][j] == 1 and i != j:
```

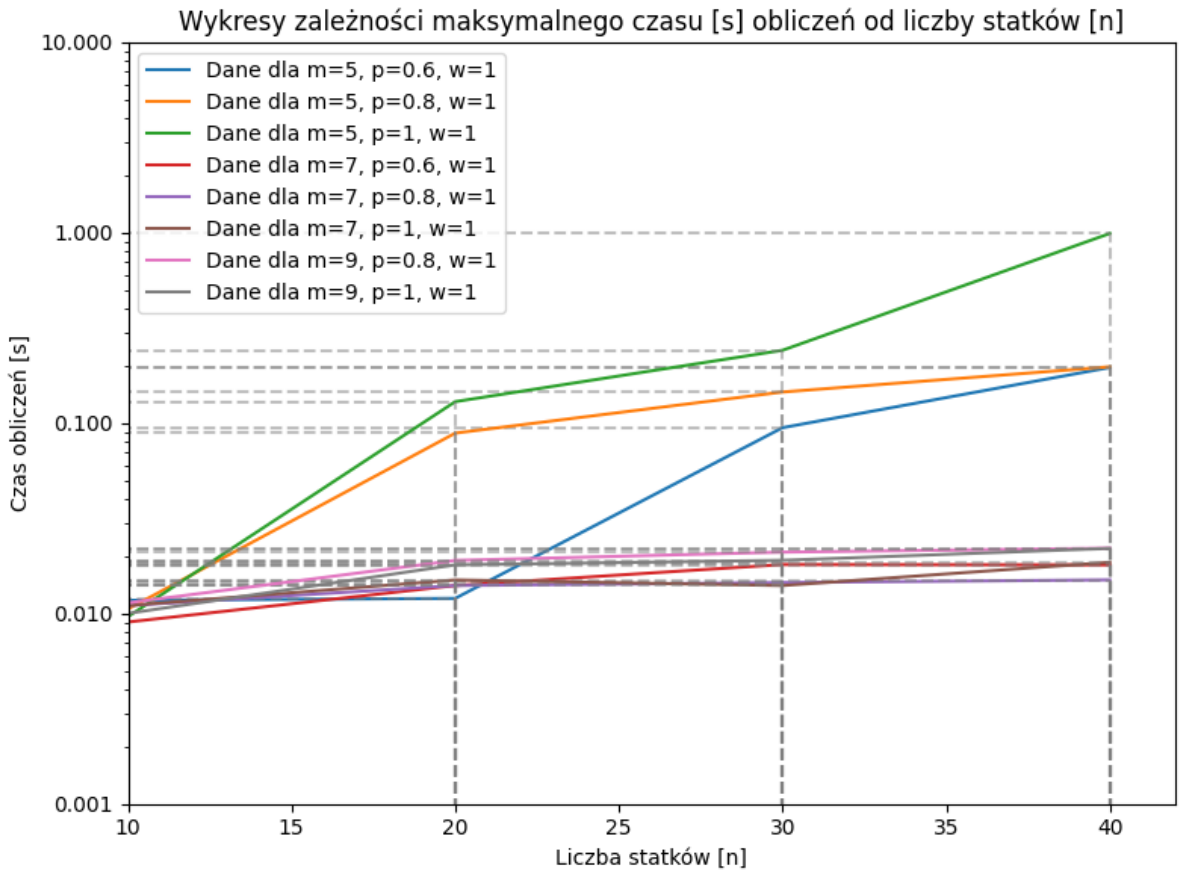
```
                                nr_pierwszego_samolotu = int(i/m)
                                nr_pierwszego_manewru = j
                            while nr_pierwszego_manewru > (m-1):
                                nr_pierwszego_manewru = nr_pierwszego_manewru - m
                                nr_drugiego_samolotu = int(j/m)
                                nr_drugiego_manewru = i
                                while nr_drugiego_manewru > (m-1):
                                    nr_drugiego_manewru = nr_drugiego_manewru - m
                                model.add_constraint(x[nr_pierwszego_samolotu,
nr_pierwszego_manewru] + x[nr_drugiego_samolotu, nr_drugiego_manewru] <= 1)
                        objective_expr = model.sum(x[i, j] * (j+1) for i in range(0, n) for j in
range(0, m))
                        model.minimize(objective_expr)
                        start = time.time()
                        solution = model.solve()
                        end = time.time()
                        print(f"Czas potrzebny na obliczenia: {end-start}[s]")
                        czasy.append(end-start)
                        if model.populate() != None:
                            print(f"Liczba znalezionych rozwiązań dopuszczalnych:
{model.populate().size}")
                            ilosc_rozw_dop.append(model.populate().size)
                        else:
                            ilosc_rozw_dop.append("brak roz dop")
                with open(pref+'samoloty.out', 'w') as file:
                    file.write("zebrane czasy: \n")
                    for czas in czasy:
                        file.write(str(czas))
                        file.write('\n')
                    file.write("zebrana liczba rozwiązań dop: \n")
                    for dop in ilosc_rozw_dop:
```

## 5.Wyniki eksperymentu.

-badanie czasu dla zmiennego n:

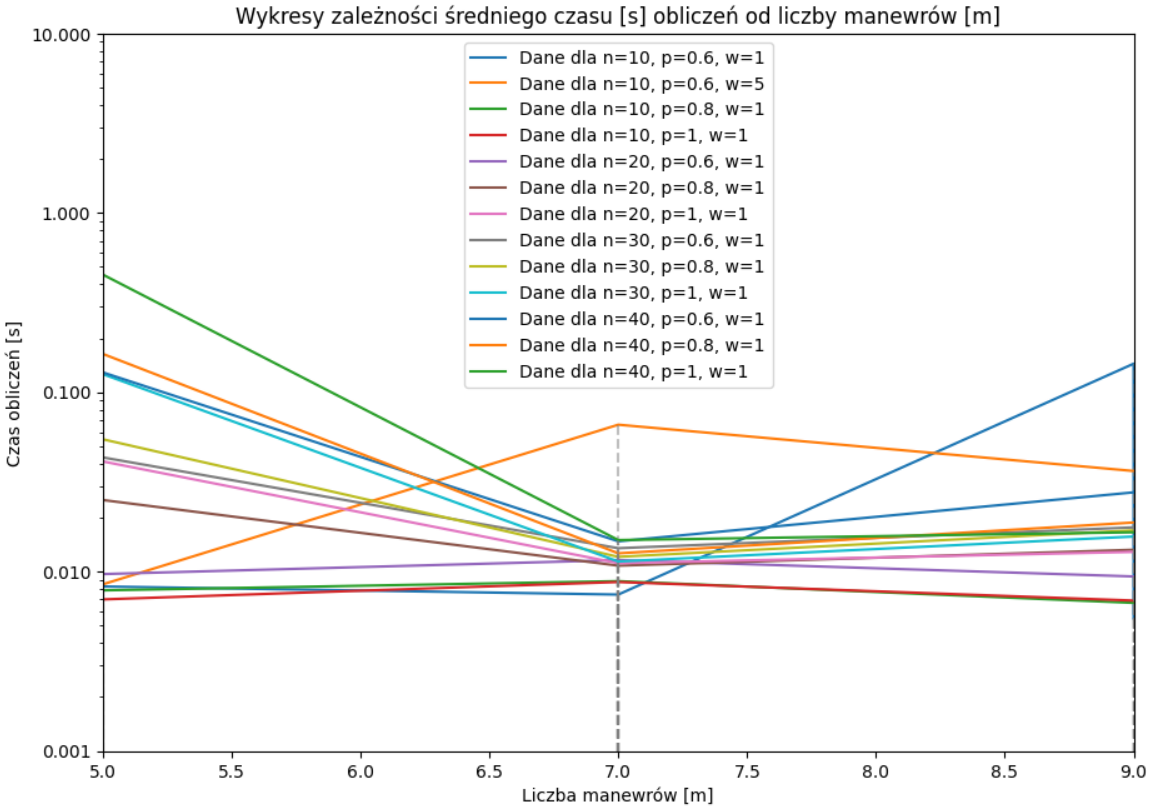


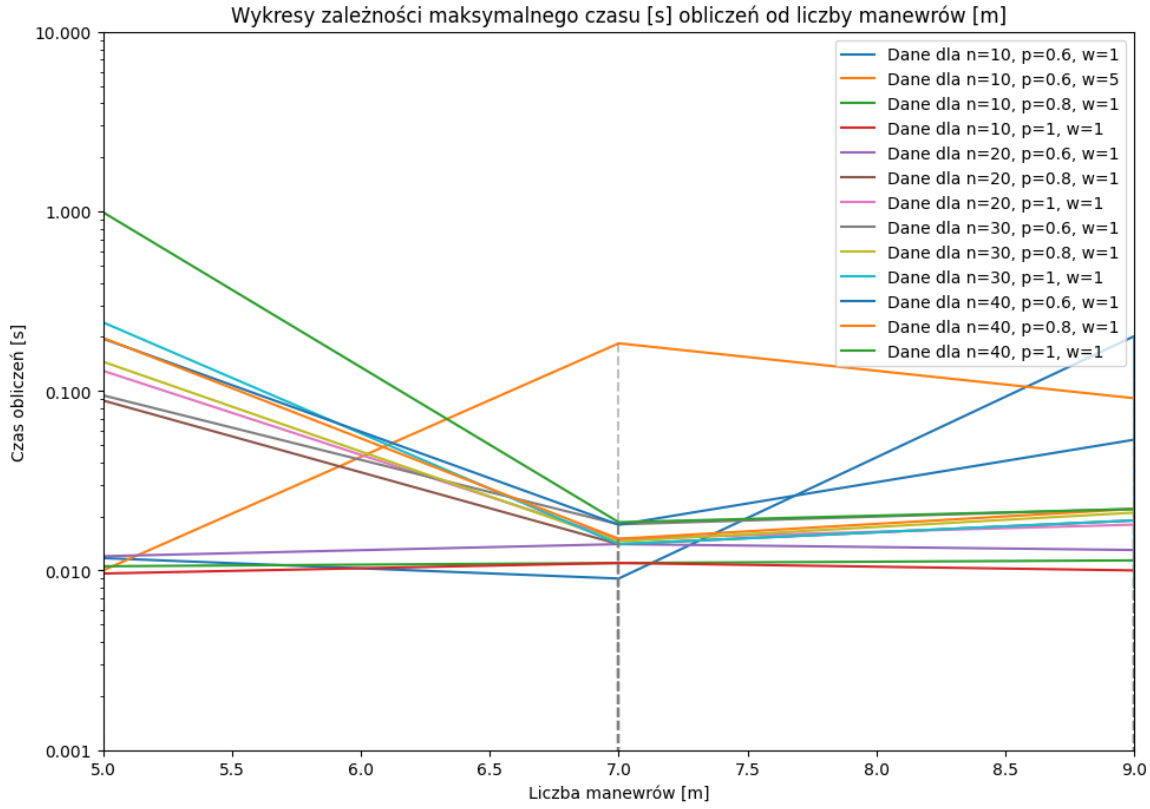



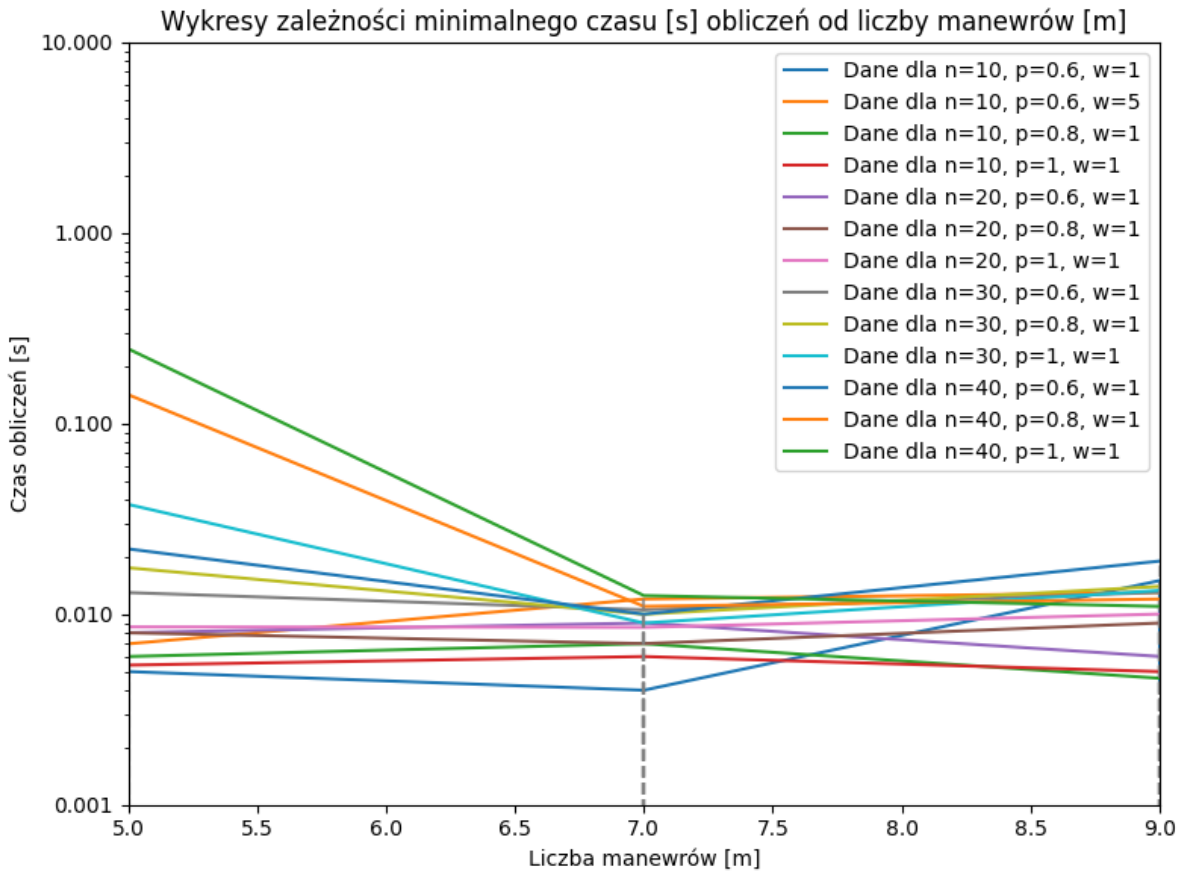



Na wszystkich 3 wykresach możemy dostrzec, że czasy obliczeń dla m = 5 znacznie wzrastają razem z większymi wartościami n, czego nie można powiedzieć o pozostałych wykresach które utrzymują mniej więcej stały poziom dla każdej wartości n. Dzieję się tak zapewne dlatego, że zwiększenie liczby dostępnych manewrów powoduję pojawienie się większej ilości opcji na znalezienie rozwiązania optymalnego. Na wykresach nie umieściłem wyników pomiaru czasu dla instancji, które miały parametr m = 3 gdyż w żadnej z nich algorytm nie odnalazł rozwiązania dopuszczalnego. Jeśli chodzi o ilość rozwiązań dopuszczalnych znalezionych dla naszych trendów to dla:
-niebieskiego mamy średnio 86,625 rozwiązań, maksymalnie mamy 130 a minimalnie 70
-pomarańczowego mamy średnio 91,5 rozwiązań, maksymalnie mamy 151 a minimalnie 61
-zielonego mamy średnio 84,575 rozwiązań, maksymalnie mamy 133 a minimalnie 66

## -badanie czasu dla zmiennego m:

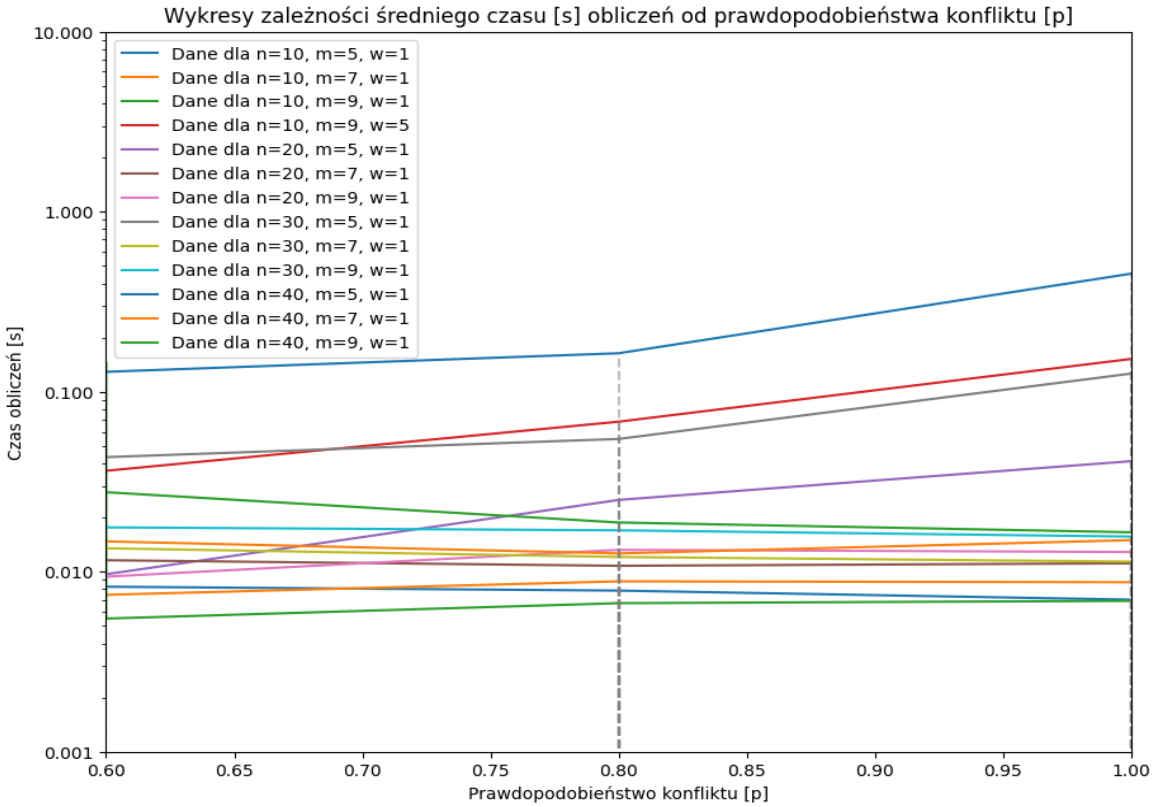


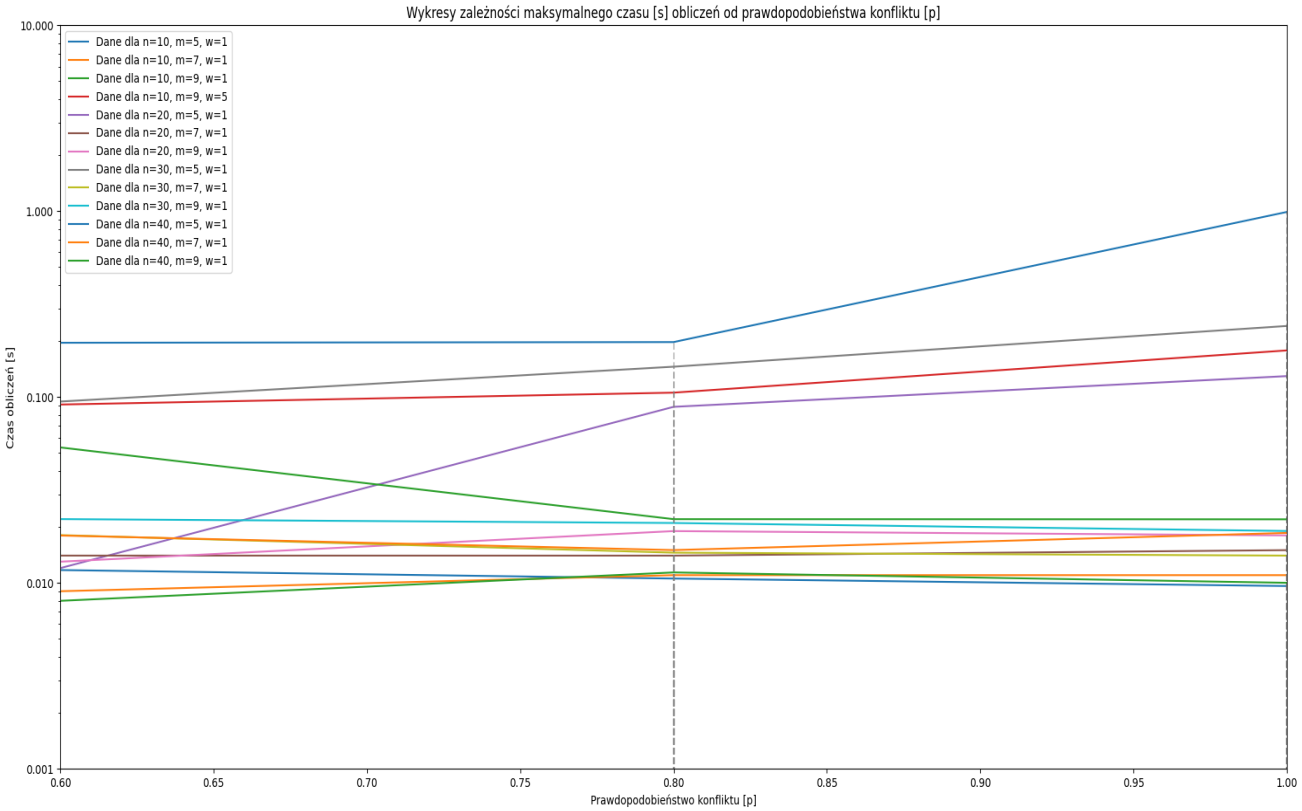



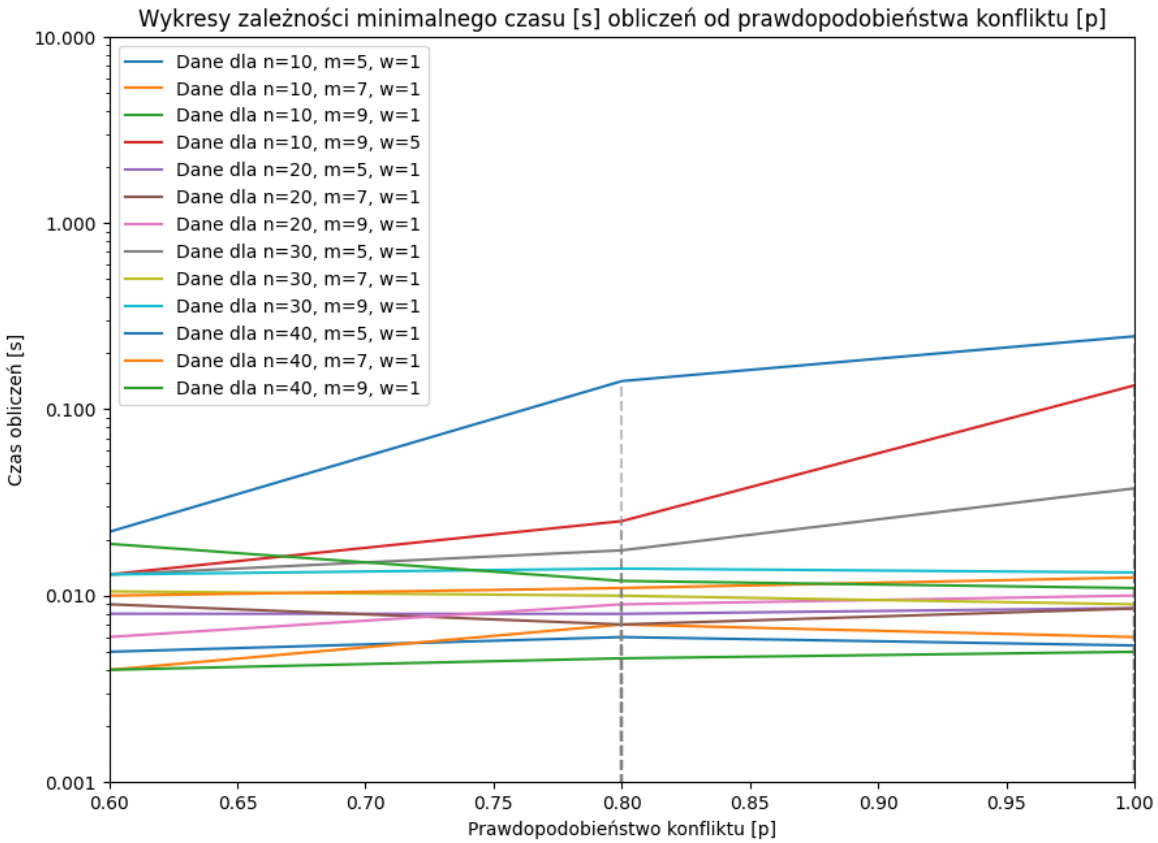



Na wszystkich trzech wykresach widzimy trend spadkowy między zmiana parametru m z 5 na 7, który później przekształca się w stały, możemy z tego wnioskować że liczba manewrów 7 jest optymalną wartością dla pozwalająca na szybkie i łatwiejsze znajdowanie rozwiązań optymalnych . Na wykresach nie zostały uwzględnione instancje z m = 3 gdyż praktycznie w żadnej z nich nie udało się znaleźć rozwiązania dopuszczalnego. Jest to spowodowane zapewne zbyt mała liczba manewrów która nie wystarcza do wyznaczenia odpowiedniej bezkolizyjnej ścieżki dla wszystkich samolotów.

## -badanie czasu dla zmiennego p:






Na wszystkich wykresach dominują trendy stałe, w niektórych przypadkach delikatnie wzrostowe. Takie wyniki dają ukazują nam raczej nie oczekiwane wyniki gdyż można się było spodziewać że wzrost prawdopodobieństwa konfliktu zmniejszy czas poszukiwania optymalnego wyniku ze względu na najmniej dostępnych optymalnych ścieżek z których algorytm mógł by wypierać. Tak jak w poprzednich przykładach brak tu danych dla instancji ze zmienną m = 3 powody są takie same jak wyżej.

**-badanie czasu dla zmiennego w:**

Jeśli przyjrzymy się wszystkim wykresom jakie zostały tu umieszczone możemy zauważyć, że praktycznie na żadnym nie ma instancji z parametrem „w" różnym od 1, jest to spowodowane tym, że we wszystkich (pomijając kilka) przypadkach gdy parametr ten był zwiększany w macierzy konfliktów pojawiało się na tyle dużo jedynek że praktycznie uniemożliwiały one odkrycie rozwiązania dopuszczalnego.

## 6. Wnioski.

Na podstawie przeprowadzonego eksperymentu, który miał na celu zbadanie efektywności algorytmu solwera CPLEX w rozwiązywaniu problemu programowania matematycznego dla wybierania optymalnych manewrów samolotów, można sformułować następujące wnioski:

**Wpływ liczby samolotów (n) na czas obliczeń:** Analizując wykresy można zauważyć że dla małej liczby manewrów m czasy dla kolejnych wartości n znacząco wzrastają, jednak gdy zwiększymy m trend z rosnącego zamienia się w stały.

**Wpływ liczby manewrów (m) na czas obliczeń**: Zauważono, że zwiększenie liczby dostępnych manewrów dla każdego samolotu (parametr m) prowadzi do zmniejszenia czasu obliczeń, szczególnie gdy m przechodzi z 5 do 7. Jest to spowodowane wzrostem liczby możliwych konfiguracji, które mogą prowadzić do rozwiązania . Warto zauważyć, że dla m=7 czas obliczeń zaczyna się stabilizować, co sugeruje, że jest to liczba manewrów, która umożliwia efektywne znalezienie rozwiązania.

**Nieznalezienie rozwiązań dla niskiej liczby manewrów (m=3):** Brak możliwości znalezienia rozwiązania dopuszczalnego dla instancji z m=3 wskazuje na niewystarczającą liczbę opcji manewrowych dla samolotów, co uniemożliwia uniknięcie konfliktów.

**Zmienne p i w oraz ich wpływ na wyniki:** Stosunkowo stały poziom czasu obliczeń przy różnych wartościach p (prawdopodobieństwo wystąpienia konfliktu) może wskazywać, że solwer jest w stanie efektywnie radzić sobie z różnymi poziomami prawdopodobieństwa konfliktów. Jednakże, zwiększanie parametru w, czyli miary „zachowania konfliktu", prowadzi do zbyt dużego nasycenia macierzy konfliktów, co czyni problem niemożliwym do rozwiązania w ramach dostępnych manewrów.

**Efektywność algorytmu CPLEX:** Algorytm CPLEX wykazał zdolność do skutecznego rozwiązywania skomplikowanych problemów programowania matematycznego w warunkach zwiększającej się złożoności. Solwer był zdolny do znajdowania rozwiązań dopuszczalnych w większości badanych przypadków, szczególnie gdy dostępna była odpowiednia liczba manewrów i parametr w = 1.