Wydział Podstawowych Problemów Techniki
Politechnika Wrocławska

Aplikacja wspomagająca morskie żeglarstwo turystyczne

Dagmara Gluch
Nr indeksu: 244758

Praca inżynierska napisana
pod kierunkiem
dr. hab. Szymona Żeberskiego

Politechnika
Wrocławska
Wrocław 2021

Spis treści

1 Wstęp **1**
- 1.1 Nakreślenie problematyki . . . 1
- 1.2 Cel pracy . . . 1
- 1.3 Zakres pracy . . . 1
- 1.4 Stosowane obecnie rozwiązania . . . 1
- 1.5 Przedstawienie poszczególnych rozdziałów pracy . . . 2

2 Analiza problemu **3**
- 2.1 Analiza teoretyczna zagadnienia . . . 3
- 2.2 Schemat rozwiązania problemu . . . 3

3 Projekt systemu **5**
- 3.1 Charakterystyka grafu . . . 5
  - 3.1.1 Wyznaczenie obszaru . . . 5
  - 3.1.2 Kształt grafu . . . 6
  - 3.1.3 Tworzenie grafu . . . 7
  - 3.1.4 Reprezentacja grafu . . . 9
- 3.2 Dane pogodowe . . . 9
- 3.3 Wagi krawędzi . . . 10
  - 3.3.1 Biegunowa prędkości . . . 10
  - 3.3.2 Dane pogodowe a wykres biegunowych prędkości . . . 11
  - 3.3.3 Długość odcinka trasy . . . 14
  - 3.3.4 Obliczenie wag . . . 14
- 3.4 Zastosowane algorytmy . . . 14
  - 3.4.1 Problem najkrótszej ścieżki . . . 14
  - 3.4.2 Podproblem I . . . 15
  - 3.4.3 Podproblem II . . . 16

4 Analiza projektowa **17**
- 4.1 Wymagania funkcjonalne . . . 17
- 4.2 Wymagania niefunkcjonalne . . . 17
- 4.3 Przypadki użycia . . . 17
- 4.4 Diagram swimlane . . . 18

5 Implementacja systemu **19**
- 5.1 Opis technologii . . . 19
- 5.2 Omówienie kodów źródłowych . . . 19

6 Instalacja i wdrożenie **21**
- 6.1 Instalacja . . . 21
- 6.2 Instrukcja i przedstawienie aplikacji . . . 21

7 Testy **25**
- 7.1 Czas przepłynięcia trasy a zmiany prognozy pogody . . . 25

7.2 Czas przepłynięcia trasy a model jachtu 27

8 Podsumowanie **29**

Bibliografia **31**

A Zawartość płyty CD **33**

Wstęp

1.1 Nakreślenie problematyki

Podczas uprawiania morskiego żeglarstwa turystycznego pojawia się problem zaplanowania trasy rejsu. Osoba planująca morski rejs turystyczny ma zazwyczaj ograniczony czas, jaki może na niego przeznaczyć - czy to ze względu na określony czas urlopu, czy też długość czarteru jachtu. W związku z tym zadaje sobie pytania, które mogą brzmieć następująco: ile czasu zajmie w tym terminie przepłynięcie z Gdyni do Sztokholmu? Czy tydzień na to wystarczy czy może trzeba myśleć o popłynięciu do innego portu? A może powinno to trwać krócej i można jeszcze gdzieś popłynąć?
Odpowiedź na te pytania nie jest prosta. Wyznaczenia w miarę dokładnego czasu pokonania określonej trasy przez jacht zależy od wielu czynników i zmiennych, wymaga też dokładnego sprawdzenia prognozy pogody. Obliczenie szukanego czasu bez narzędzi informatycznych jest trudne i nieprecyzyjne.

1.2 Cel pracy

Celem pracy było stworzenie aplikacji dla osób uprawiających morskie żeglarstwo turystyczne. Główna użyteczność to obliczanie czasu, w jakim morski jacht żaglowy pokona określoną trasę uwzględniając prognozę pogody.
Aplikacja ta na podstawie danych wprowadzonych przez użytkownika takich jak: port startowy, port końcowy, planowana data wypłynięcia oraz model jachtu ma wyznaczać optymalną (pod względem czasu) trasę oraz obliczać czas jej pokonania dla obszaru Morza Bałtyckiego.

1.3 Zakres pracy

Zakres pracy obejmuje:

- pozyskanie i odpowiednie przetworzenie danych uzyskanych na podstawie prognozy pogody, map geograficznych oraz wykresu biegunowej jachtu
- stworzenie i wykorzystanie grafu ważonego do reprezentacji tych danych
- analizę i implementację algorytmów znajdywania najkrótszej ścieżki w oparciu o uzyskany graf
- stworzenie aplikacji prezentującej obliczone dane w czytelny sposób

1.4 Stosowane obecnie rozwiązania

Obecnie nie są dostępne aplikacje zajmujące się zagadnieniem obliczenia czasu, w jakim morski jacht żaglowy pokona daną trasę. Problem ten jest najczęściej rozwiązywany poprzez szacowanie za pomocą iloczynu średniej prędkości danego modelu jachtu i długości szukanej trasy. Rozwiązanie to jest obarczone dużą niedokładnością oraz niewygodne – wymaga znalezienia długości trasy, samodzielnego wykonania obliczeń, a

zainteresowany dodatkowo musi samodzielnie sprawdzić prognozę pogody, aby orientacyjnie ją uwzględnić.

1.5 Przedstawienie poszczególnych rozdziałów pracy

Praca składa się z ośmiu rozdziałów. Kolejny, po wstępie, rozdział przybliża teoretyczne aspekty zagadnienia i dokładniej wprowadza w tematykę pracy. W rozdziale trzecim szczegółowo przedstawiono sposób rozwiązania zadanego problemu. Porusza on kolejne etapy pracy i aktywności zachodzące w programie, opisuje wykorzystane pomysły i algorytmy. Aspekty projektowe systemu są zawarte w rozdziale czwartym. W rozdziale piątym zostały opisane technologie użyte do stworzenia aplikacji. W rozdziale szóstym przedstawiono sposób instalacji i wdrożenia systemu. Rozdział siódmy zawiera testy i przedstawienie aplikacji. Ostatni rozdział jest podsumowaniem pracy.

Analiza problemu

2.1 Analiza teoretyczna zagadnienia

Głównym problemem pracy było obliczenie czasu przepłynięcia przez jacht określonej trasy. Dla jachtów żaglowych czas ten zależy od bardzo wielu parametrów.

Parametry mające największy wpływ to:

- długość trasy
- siła wiatru działająca na jacht
- kształt jachtu
- użycie silnika

Główną trudnością było uwzględnienie tych, bardzo różnych parametrów. Dodatkowo dane na temat wiatru z danej godziny są pobierane z prognozy pogody, a więc są (tak jak sama prognoza) zmienne w czasie. Oprócz tego wiatr jest wielkością wektorową. Należało więc uwzględnić nie tylko jego siłę, lecz także kierunek i zwrot (a dokładniej istotny jest kąt między wektorem wiatru a osią jachtu).

Kolejnym problemem był fakt, że wiatr o danej sile, działający pod danym kątem działa w różny sposób na różne modele jachtów. Każdy model jachtu ma inną masę, kształt, powierzchnię ożeglowania, inaczej zachowuje się przy różnych prędkościach, kątach przechylenia, ma inny opór boczny, opór na sterze itp. (podobnie jak ma to miejsce przy różnych modelach samochodów). W celu przeliczenia tych parametrów na chwilową prędkość jachtu został wykorzystany *wykres biegunowych prędkości jachtu*, który łączy dane o wpływie wiatru na prędkość danego modelu jachtu.

Jeśli chodzi o użycie silnika, które w oczywisty sposób wpływa na prędkość jachtu, a tym samym na czas pokonania danej trasy to praca ta zakłada, że silnik nie jest używany. Jest to uzasadnione różnymi względami: głównym celem tej pracy jest obliczenie czasu pokonania trasy przez jacht żaglowy i uwzględnienie parametrów wpływających właśnie na ruch pod żaglami, co jest głównym urokiem tego sportu. Silnik oczywiście jest również używany, ale korzystanie jednocześnie z siły napędu za pomocą żagli i silnika nie jest dobrze postrzegane w środowisku żeglarskim, jest wręcz określane w pogardliwy sposób.

Informacje i terminologia żeglarska użyta w pracy pochodzi głównie z książki [4].

2.2 Schemat rozwiązania problemu

1. Dany akwen (morze) jest reprezentowany jako graf
2. Z prognozy pogody otrzymywane są dane o prędkości i kierunku wiatru
3. Na ich postawie jest obliczany czas, jaki zajmie jachtowi pokonanie danego odcinka trasy
4. Koszt ten jest reprezentowany jako wagi krawędzi grafu

5. Używając algorytmów znajdowania najkrótszej ścieżki jest wyznaczana optymalna trasa i obliczony czas jej pokonania

Projekt systemu

3.1 Charakterystyka grafu

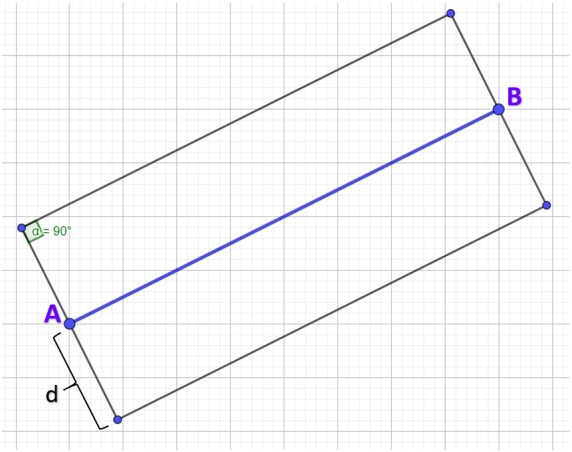
Pierwszym etapem rozwiązania problemu jest stworzenie grafu, który będzie reprezentował dany akwenu. Graf musiał spełniać wiele różnych funkcji:

- przechowywać informacje o porcie startowym i końcowym
- odzwierciedlać odpowiedni fragment obszaru Morza Bałtyckiego
- przechowywać dane o pogodzie, w tym różnice między wpływem poszczególnych kątów między wektorem wiatru a osią jachtu
- przechowywać dane o sąsiedztwie wierzchołków
- być zareprezentowany w sposób, który będzie wygodny w użyciu w algorytmach wyszukiwania najkrótszej ścieżki

3.1.1 Wyznaczenie obszaru

Obszar morza, który będzie uwzględniony w obliczeniach, a więc zareprezentowany za pomocą grafu jest wyznaczany na podstawie otrzymanych danych o porcie startowym i końcowym (dalej odpowiednio punkty A i B). Określenie ograniczonego obszaru jest konieczne, ponieważ uwzględnienie w obliczeniach całego Morza Bałtyckiego byłoby zbędne i nieefektywne.

Na początku wprowadzone zostają współrzędne geograficzne punktów A i B. Następnie na ich podstawie wyznaczany jest obszar, który ma kształt prostokąta, gdzie odcinek $|AB|$ jest jego środkową. Długość d zaznaczona na rysunku 3.1 ma 0,5 stopnia geograficznego. Obliczane są współrzędne punktów będące wierzchołkami uwzględnianego obszaru.



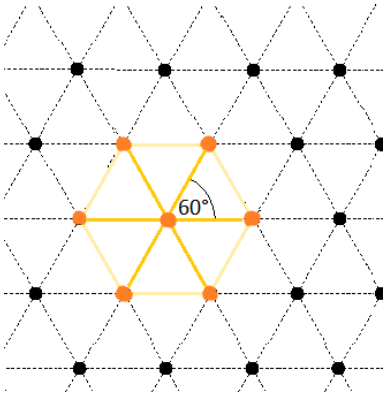
Rysunek 3.1: Schemat kształtu obszaru

3.1.2 Kształt grafu

W tym podrozdziale zostanie poruszona kwestia, w jaki sposób należy zbudować i połączyć z sobą sieć wierzchołków grafu (podobnie jak ma to miejsce przy topologii sieci komputerowej). Zostało to określone dalej jako kształt grafu.

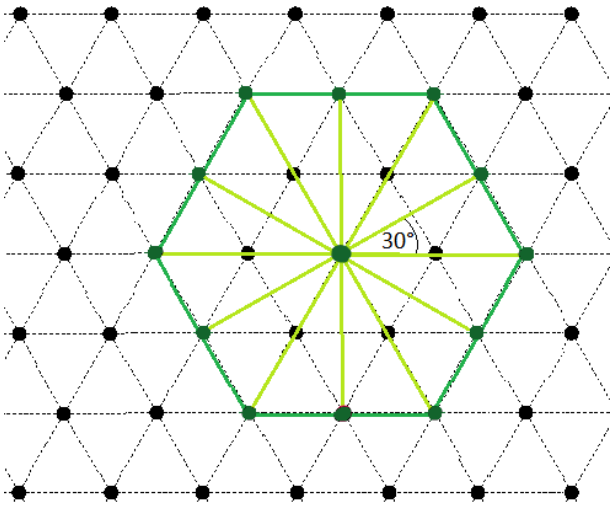
Ponieważ wierzchołki grafu odzwierciedlają punkty w terenie konieczne było ich regularne rozmieszczenie. Z tego powodu w pracy zastosowany został graf w kształcie kraty trójkątnej. Wybór kraty trójkątnej był związany z potrzebą odwzorowania różnych, możliwie wielu kierunków wiatru, co nie byłoby możliwe przy użyciu na przykład kraty kwadratowej, przy czym pojęcie *kierunek wiatru* rozumiane jest tutaj jak w terminologii żeglarskiej (a nie ściśle fizycznej) - jako kąt pomiędzy wektorem wiatru a osią jachtu.

Samo zastosowanie grafu w kształcie kraty trójkątnej nie rozwiązuje jednak wciąż powyższego problemu. W grafie o takim kształcie intuicyjnie sąsiedzi danego wierzchołka powinni leżeć na wierzchołkach sześciokąta, jak zaznaczono na rysunku 3.2. Wtedy jednak kąt między dwoma sąsiednimi krawędziami wynosiłby 60°. Nie daje to zadowalającej dokładności - między kierunkami wiatru różniącymi się o 60° są zbyt duże różnice we wpływie działania siły na jacht.



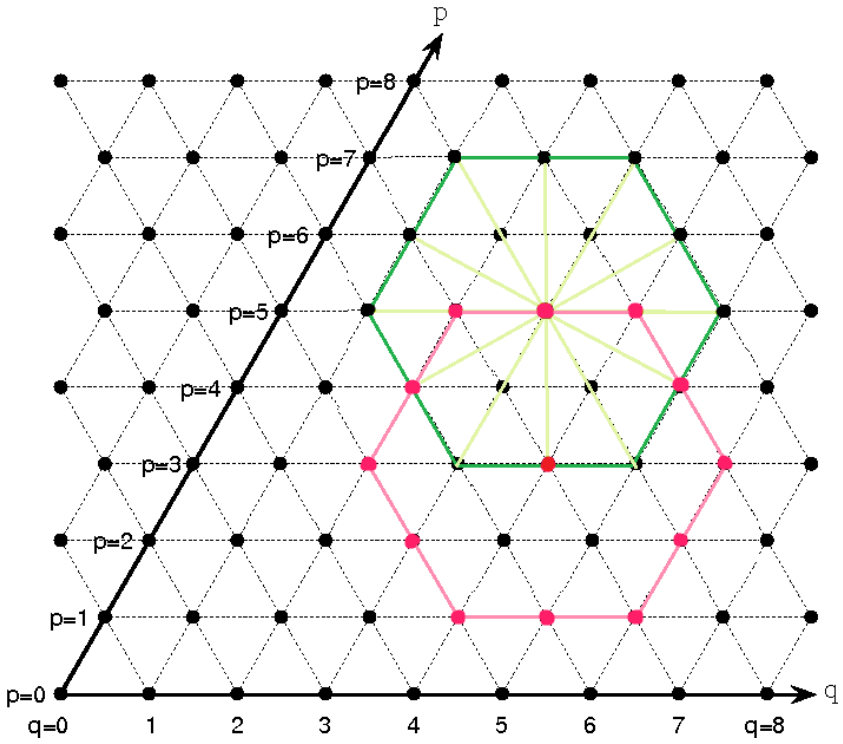
Rysunek 3.2: Wierzchołek i jego 6 sąsiadów - wersja pierwotna

Potrzebna była większa dokładność, a więc mniejszy kąt i więcej krawędzi wychodzących z jednego wierzchołka. Dlatego zastosowany został "skok o 2" w sześciokącie - wierzchołki sąsiadujące zostały rozmieszczone tak jak to pokazuje rysunek 3.3, co dało kąt 30° między sąsiednimi krawędziami, a w konsekwencji dostateczną dokładność.



Rysunek 3.3: Wierzchołek i jego 12 sąsiadów - wersja zastosowana

Konstruowanie kolejnych wierzchołków i ich sąsiadów odbywa się na tej samej zasadzie. Ilustruje to rysunek 3.4. Kolejny rozpatrywany wierzchołek i jego sąsiedzi zostali oznaczeni kolorem różowym.



Rysunek 3.4: Konstrukcja kolejnego wierzchołka i jego sąsiadów

3.1.3 Tworzenie grafu

Graf $G = (V, E)$ jest tworzony na podstawie współrzędnych geograficznych wprowadzonych punktów A i B odpowiadających portom startowemu i końcowemu. W algorytmie używane są:

- *area* oznaczająca rozpatrywany obszar, który będzie zareprezentowany za pomocą grafu,
- S - stos przechowujący punkty, które jeszcze należy sprawdzić,
- *vectors* - tablica zawierająca 12 wektorów, na podstawie których ze współrzędnych danego wierzchołka (punktu) obliczane są współrzędne jego potencjalnych 12 sąsiadów,
- V - zbiór wszystkich wierzchołków grafu G; na początku wykonywania algorytmu jest pusty - kolejne wierzchołki są sukcesywnie dodawane,
- G - konstruowany graf składający się docelowo z $|V|$ list sąsiedztwa $Adj[v]$ dla każdego $v \in V$.

Używane zmienne: p - aktualnie rozpatrywany punkt, n aktualnie rozpatrywany sąsiad. W punktach x oznacza pierwszą współrzędną punktu, a y drugą.

Przed rozpoczęciem działania algorytmu, na podstawie punktów A i B zostaje wyznaczone *area* w sposób opisany w podpunkcie 3.2.1.

Pseudokod 3.1: Algorytm tworzenia grafu

```
   Input: punkty A i B
   Output: graf G
 1 S.push(A)
 2 while S ≠ ∅ do
 3     p ← S.pop()
 4     for i ← 1 to vectors.lenght do
 5         n ← (p.x + vectors[i].y, p.y + vectors[i].x)
 6         if areaContains(n) then
 7             if isPointInVertices(n) == false then
 8                 if isWater(n) == true then
 9                     V.add(n)
10                     S.push(n)
11                     G.add(Adj[n])
12             G.get(Adj[p]).add(n)
13 addEndPointToGraph(B)
```

W wierszu 1 do stosu S dodawany jest punkt A, następnie dopóki S nie jest pusty sprawdzane są punkty, które potencjalnie mogą należeć do grafu. Polega na ściągnięciu danego punktu p ze stosu (wiersz 3) i wyliczeniu współrzędnych 12 punktów n - potencjalnych wierzchołków sąsiadujących z p (wiersze 4-5). Przy obliczaniu do pierwszej współrzędnej punktu p jest dodawana druga współrzędna wektora, a to drugiej współrzędnej p pierwsza współrzędna wektora. Wynika to z tego, że w punktach geograficznych pierwsza współrzędna oznacza szerokość geograficzną (a więc przesunięcie w pionie), a druga długość geograficzną (przesunięcie w poziomie) - odwrotnie niż ma to miejsce w przypadku kartezjańskiego układu współrzędnych. Następnie ma miejsce sprawdzenie, czy dany punkt n można dodać do grafu, a więc czy należy do rozpatrywanego obszaru *area* i czy reprezentuje morze (`isWater`). W zależności od tego, czy n znajduje się już w zbiorze wierzchołków, czy też nie, jest on dodawany do odpowiednich struktur danych. Na końcu do grafu dodawany jest punkt B (wiersz 13).

Realizacja funkcji `isPointInVertices` sprawdzającej, czy dany punkt jest już w zbiorze V jest specyficzna. Funkcja ta jako parametr przyjmuje punkt q, a następnie iteruje po zbiorze V sprawdzając, czy zbiór już taki punkt zawiera, a więc czy w zbiorze V istnieje punkt p przystający do punktu q. Pod względem matematycznym dwa punkty są przystające kiedy wszystkie ich współrzędne są sobie parami równe. Warunek sprawdzający równość byłby więc zapisany w sposób

$$x_p = x_q \quad AND \quad y_p = y_q$$

Ponieważ dokładność obliczeń komputerowych jest ograniczona, a przy obliczaniu wartości współrzędnych punktów jest wykorzystywana liczba niewymierna $\sqrt{3}$ (to wynika z zależności trygonometrycznych zachodzących w trójkącie równobocznym), to w pewnym momencie dokładne, matematyczne porównanie dwóch współrzędnych staje się niemożliwe i nie spełnia swojej funkcji. W celu sprawdzenia "równości" dwóch punktów należy je więc porównać co do pewnej dokładności ε. Warunek sprawdzający wygląda więc następująco:

$$|x_p - x_q| < \varepsilon \quad AND \quad |y_p - y_q| < \varepsilon$$

Ponieważ rozpatrywane punkty (później wierzchołki w grafie G) reprezentują nie tylko jeden dokładny punkt w terenie, ale także cały obszar wokół nich, który jest do nich sprowadzany, to jest to rozwiązanie lepsze. Dokładność ε została ustalona jako długość boku trójkąta równobocznego stanowiącego fragment kraty trójkątnej.

Funkcja `isWater(`p`)` wykonuje zapytanie do API i na podstawie współrzędnych punktu p zwraca `true` jeśli dany punkt jest morzem, `false` w przeciwnym razie. Pozwala to na modelowanie trasy jachtu z ominięciem wysp.

Naniesienie punktu końcowego

Po stworzeniu grafu należy nanieść na niego wierzchołek reprezentujący port końcowy - punkt B. Robi to funkcja `addEndPointToGraph`. Najpierw przeglądany jest zbiór wszystkich wierzchołków V, z którego jest wybierane 6 wierzchołków leżących najbliżej punktu B, które są łączone z B. Dla każdego z nich tworzona jest nowa krawędź (v, B) i dodawana do listy sąsiedztwa danego wierzchołka v. Z wierzchołka B nie wychodzą już żadne krawędzie, jest on dodawany jedynie do listy wszystkich wierzchołków. W implementacji konieczne było przechowanie nie tylko informacji o tym, jakie wierzchołki łączy dana krawędź, ale musi ona też reprezentować kurs rzeczywisty, jakim porusza się jacht, a więc kąt między północą (0°) a osią jachtu (z dokładnością do 30°). Można to zobrazować pytaniem, jaki kąt na tarczy zegara tworzy o pełnych godzinach wskazówka godzinowa z minutową, przy czym zakres kątów to $\langle 0°, 360° \rangle$.

Obliczenie tego kąta na postawie samych współrzędnych geograficznych punktów jest podobne do obliczania kąta między osią układu współrzędnych a prostą, tam jednak zakres to $\langle 0°, 180° \rangle$.

Dany jest punkt $B = (x_b, y_b)$ i wierzchołek v oznaczany jako punkt $W = (x_w, y_w)$. Niech a - współczynnik kierunkowy prostej, w której zawiera się odcinek $|WB|$, φ - kąt między prostą a osią Ox, ψ - szukany kąt. Korzystając ze wzoru na współczynnik kierunkowy prostej przechodzącej przez dwa punkty otrzymano:

$$a = tg(\varphi) = \frac{y_b - y_w}{x_b - x_w} \implies \varphi = arctg(a)$$

Ponieważ konieczne jest obliczenie kąta między osią Oy a prostą oraz ze względu na zakres $\langle 0°, 360° \rangle$, wartość kąta ψ wyraża się wzorem:

$$\psi = \begin{cases} 90° - \varphi & \text{gdy } x_w \leqslant x_b \text{ (kursy } 0° - 180°) \\ 180° + 90° - \varphi & \text{gdy } x_w > x_b \text{ (kursy } 180° - 360°) \end{cases}$$

Po obliczeniu wartości jest ona przybliżana do najbliższej wielokrotności 30° - jest to dokładność, którą umożliwia kształt grafu (rysunek 3.3).

3.1.4 Reprezentacja grafu

Istnieją dwa standardowe sposoby reprezentacji grafu: za pomocą list sąsiedztwa lub macierzy sąsiedztwa (dokładny opis w [1, rozdział 22.1]). Pierwszy jest zazwyczaj stosowany do reprezentacji grafów rzadkich, drugi gęstych. Zgodnie z definicją graf $G = (V, E)$, gdzie V oznacza zbiór wierzchołków, a E zbiór krawędzi graf jest rzadki, kiedy $|E|$ jest dużo mniejsze od $|V|^2$.

Wybór reprezentacji grafu był konsekwencją opisanego wyżej kształtu grafu. W zastosowanym grafie z każdego wierzchołka wychodzi 12 krawędzi, więc $|E| = 12|V|$ Należy zauważyć, że zgodnie z przywołaną definicją gęstości użyty w tej pracy graf jest rzadki dla $|V| > 12$. Biorąc pod uwagę realne przypadki grafów konstruowanych podczas działania programu można przyjąć, że jest rzadki zawsze.

Zastosowana została więc reprezentacja listowa. Graf $G = (V, E)$ jest więc reprezentowany za pomocą tablicy zawierającej $|V|$ list, po jednej dla każdego wierzchołka z V. Dla każdego wierzchołka $u \in V$ odpowiednia lista zawiera wszystkie wierzchołki v takie, że krawędź $(u, v) \in E$ wraz z wagą krawędzi (u, v). Przykład grafu utworzonego za pomocą programu dla trasy Karlskrona - Świnoujście zareprezentowanego za pomocą list sąsiedztwa pokazuje rysunek 3.5.

Wybrana reprezentacja za pomocą list sąsiedztwa w konsekwencji dodatkową zaletę - w zastosowanym w pracy algorytmie Dijkstry konieczne jest przeglądanie sąsiadów danego wierzchołka, co zostało znacznie ułatwione.

3.2 Dane pogodowe

Drugim etapem jest pozyskanie danych z API pogodowego. W pracy zostało wykorzystane API dostarczane przez firmę Storm Glass. Dane są pobierane z okresu czasu od wprowadzonej przez użytkownika daty

```
****  Karlskrona-Świnoujście  ****

FINAL GRAPH:
0 --> 1 2
1 --> 17 3 2 0
2 --> 0 1 3 4 5
3 --> 1 17 16 4 2
4 --> 5 2 3 16 9 6
5 --> 2 4 6 7 8
6 --> 7 5 4 9 10
7 --> 8 5 6
8 --> 5 7
9 --> 4 16 12 11 10 6
10 --> 6 9 11 19
11 --> 10 9 12 13 14 19
12 --> 9 16 13 11 19
13 --> 11 12 15 14 19
14 --> 10 11 13 15 19
15 --> 14 13 19
16 --> 4 3 12 9
17 --> 1 18 3
18 --> 17 1
19 -->
```

Rysunek 3.5: Przykład stworzonego grafu dla trasy Karlskrona-Świnoujście w reprezentacji listowej

do końca zakresu pogody (pełen możliwy zakres to okres 10 dni licząc od aktualnej daty). Co ważne, dane pobrane z API są traktowane jako dane pewne. Oznacza to, że jeśli w rzeczywistości siły wiatrów w szukanym okresie będą odbiegać od prognozowanych i w związku z tym obliczona wartość czasu przepłynięcia trasy będzie znacząco odbiegać od rzeczywistej, nie jest to winą aplikacji, lecz jest to błąd prognozy.

Dane fetch'owane z API są w formacie JSON. Są one odpowiednio parsowane za pomocą biblioteki GSON, a następnie zapisywane do obiektów reprezentujących wierzchołki w postaci tablic, gdzie każda kolejna komórka przechowuje dane z kolejnej godziny prognozy.

3.3 Wagi krawędzi

Kolejnym etapem jest obliczenie wag krawędzi i nałożenie ich na stworzony graf. Waga krawędzi (u, v) wyraża szacowany czas (koszt), w jakim jacht powinien przebyć odcinek w terenie reprezentowany przez krawędź (u, v).

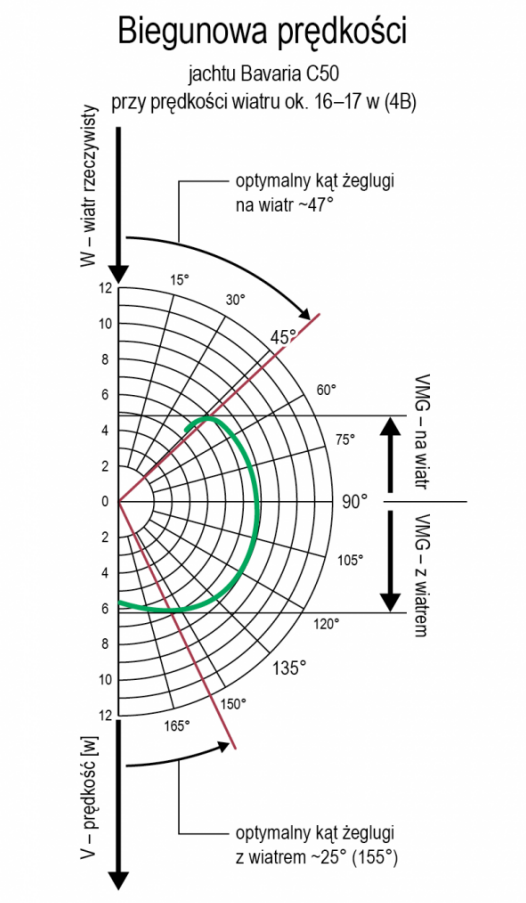
Waga krawędzi uwzględnia następujące parametry:

- kierunek wiatru
- prędkość wiatru
- model jachtu
- długość odcinka trasy

3.3.1 Biegunowa prędkości

Jak zostało wspomniane wcześniej, wpływ konkretnej siły wiatru na dany model jachtu bardzo różni się w zależności od następujących elementów: kierunku i prędkości wiatru oraz modelu jachtu, a jego obliczenie jest niełatwe. Powiązanie tych trzech parametrów ilustruje wykres biegunowych prędkości jachtu. Przedstawia on zależność prędkości jachtu od siły (prędkości) i kierunku wiatru. Przykładowy taki wykres przedstawia rysunek 3.6. Oś pionowa przedstawia prędkość wiatru w węzłach. Współrzędne biegunowe i zaznaczone kąty ilustrują kierunek wiatru - kąt, pod jakim wiatr działa na jacht (względem osi jachtu). Zielony wykres oznacza prędkość jachtu osiąganą przez jacht dla wiatru działającego pod danym kątem i o prędkości 16-17 węzłów. Przelicznik: $1w. = 1\frac{mila\_morska}{h} = 1{,}852\frac{km}{h}$. W tej pracy dla każdego modelu jachtu zostało wykorzystane

kilka wykresów biegunowych - obrazujących prędkość danego modelu jachtu względem różnych prędkości wiatru. Przedstawia to rysunek 3.7

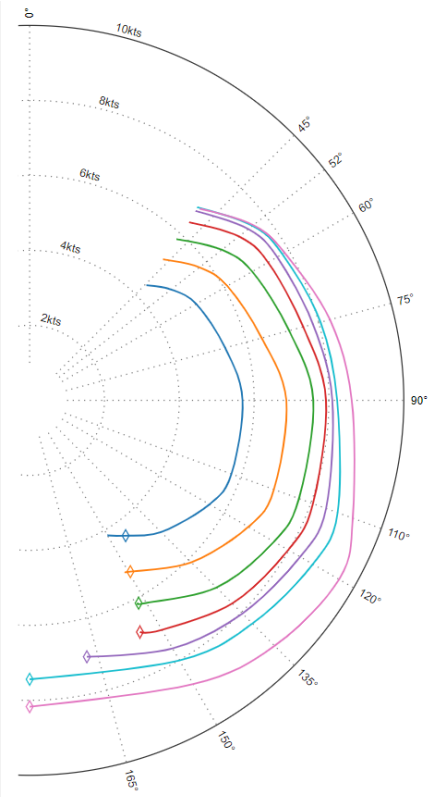


Rysunek 3.6: Biegunowa prędkości jachtu Bavaria C50

3.3.2 Dane pogodowe a wykres biegunowych prędkości

Odczytanie wartości prędkości jachtu z wykresu biegunowej na podstawie otrzymanych danych z API pogodowego nie jest takie proste. Dane pobierane z API to prędkość i kierunek wiatru. Prędkość wiatru pobrana z API dokładnie oznacza prędkość wiatru na wysokości 10 m nad poziomem morza i jest wyrażona w metrach na sekundę. Należało więc najpierw uzgodnić jednostki i przeliczyć $\frac{m}{s}$ na węzły. Dodatkowo prędkość wiatru pobrana z API ma dokładność do $0{,}1\frac{m}{s}$, natomiast wykorzystane wykresy biegunowych dostarczają danych tylko dla wybranych prędkości wiatru, jak widać na przykładzie rysunku 3.7 dla jachtu Bavaria 40 Cruiser były to dane dla: 6, 8, 10, 12, 14, 16 i 20 węzłów (powyżej 20 węzłów kształt krzywej się nie zmienia). Konieczna była kwantyzacja wartości pobranych z API. Na przykład dla wspomnianego wykresu wartość prędkości wiatru równa 5,5 węzła jest reprezentowana jako 6 w., 17 w. jako 16, a wartości większe od 18 w. jako 20 w.

Pobrana z API wartość kierunku wiatru oznacza kierunek wiatru na wysokości 10 m nad poziomem morza wyrażony w stopniach kątowych, gdzie 0° oznacza wiatr wiejący z północy. Biegunowa prędkości też używa stopni, nie ma więc potrzeby przeliczania jednostek wartości pobranych z API i użytych na wykresie. Konieczne jest jednak zauważenie, że te same jednostki wyrażają co innego - kąt użyty w wykresie biegunowej prędkości oznacza kierunek wiatru względem osi jachtu, a nie względem północy. Należało więc uzgodnić kąty.



BAVARIA 40 CRUISER

Wind velocity	6kts	8kts	10kts	12kts	14kts	16kts	20kts
Beat angles	45.5°	43.5°	42.4°	41.9°	41.3°	41°	41.7°
Beat VMG	3.07	3.76	4.3	4.75	5.05	5.15	5.11
52°	4.8	5.76	6.52	7.1	7.42	7.56	7.59
60°	5.14	6.1	6.87	7.36	7.64	7.8	7.87
75°	5.41	6.42	7.18	7.57	7.84	8.05	8.31
90°	5.69	6.86	7.58	7.92	8.09	8.22	8.61
110°	5.74	6.95	7.66	8.06	8.43	8.76	9.19
120°	5.55	6.75	7.54	7.98	8.35	8.74	9.55
135°	4.97	6.11	7.06	7.66	8.04	8.42	9.26
150°	4.16	5.28	6.24	7.07	7.64	8.01	8.74
Run VMG	3.61	4.57	5.41	6.18	6.84	7.44	8.17
Gybe angles	144.6°	149.5°	151.7°	154.5°	167.4°	180°	180°

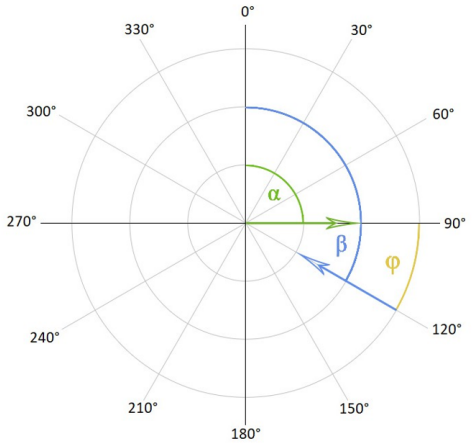
Rysunek 3.7: Biegunowe prędkości z tabelą wartości dla jachtu Bavaria 40 Cruiser

Uzgodnienie kątów

Niech α oznacza kurs, którym porusza się jacht względem północy, np. $\alpha = 90°$ to kurs na wschód. Niech β oznacza kierunek wiatru względem północy, np. $\beta = 90°$ to wiatr wiejący ze wschodu. Niech φ będzie kątem pod jakim wiatr działa na jacht, np. $\varphi = 90°$ to wiatr działający na jacht pod kątem 90° (w terminologi żeglarskiej jest to półwiatr prawego halsu). Wiatr o danej sile, wiejący pod tym samym kątem z prawej i lewej burty (strony jachtu) powoduje tę samą prędkość jachtu, np. dla $\varphi = 30°$ (wiatr działający na jacht pod kątem 30° z prawej burty) wartość prędkości jachtu będzie taka sama jak dla $\varphi = 330°$ (wiatr działający pod kątem 30° z lewej burty). Z tego powodu przyjęto oznaczanie kąta φ w zakresie $\langle 0° - 180° \rangle$. Tak więc wspomniane $\varphi = 330°$ będzie traktowane jako $\varphi = 30°$ bez określania, z której burty wieje wiatr. Przykłady obliczania kąta φ ilustrują rysunki 3.8, 3.9 i 3.10 .

Dokładność danych z wykresów biegunowych

Pomimo profesjonalnego wykonania wykresów biegunowych prędkości jachtów (większość użytych w pracy wykresów pochodzi z bazy danych ORC Sailboat Data [5] utrzymywanej przez międzynarodową organizację Offshore Racing Congress, która zajmuje się morskim żeglarstwem wyczynowym i jest odpowiedzialna za ustanawianie norm klasyfikacyjnych) ich użycie w pracy jest obarczone dużą niedokładnością - często dla jednego modelu jachtu wykresy biegunowych są różne dla jednakowych prędkości wiatru. Wynika to z faktu, że przy tej samej samej prędkości wiatru w podstawie, ale różnych porywach wartości będą inne. Oprócz tego różnice będę zależne od ułożenia żagli, a także użycia (lub nie) dodatkowych żagli (np. spinaker, genua).

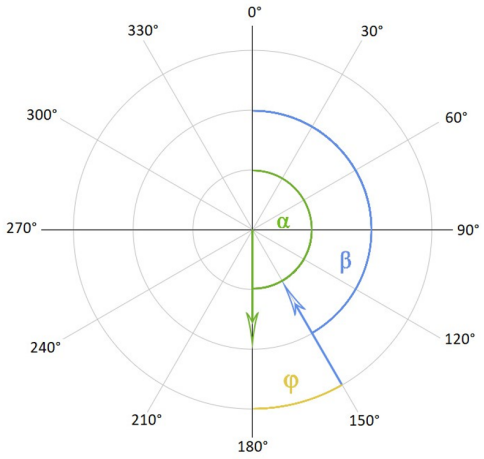


α = 90˚

β = 120 ˚

φ = 30 ˚

Rysunek 3.8: Przykład 1

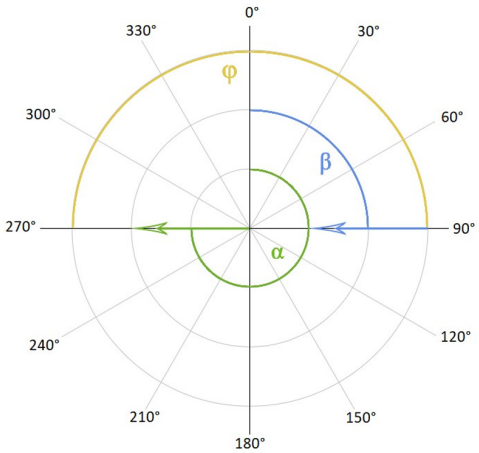


α = 180˚

β = 150 ˚

φ = 30 ˚

Rysunek 3.9: Przykład 2



α = 270˚

β = 90 ˚

φ = 180 ˚

Rysunek 3.10: Przykład 3

Przechowywanie danych z wykresów

Dane odczytane z wykresów biegunowych prędkości są umieszczone w pliach w formacie CSV i załączone do projektu (dla każdego modelu dane są umieszczone w osobnym pliku). Podczas wykonywania programu wybierany jest odpowiedni plik, a dane z niego są parsowane do *Java Object*. Rozwiązanie to sprawia, że użytkownik może w łatwy sposób dodać plik reprezentujący jego jacht.

3.3.3 Długość odcinka trasy

Obliczenie długości odcinka trasy, jaki reprezentuje dana krawędź grafu polega na na obliczeniu odległości między wierzchołkami u i v krawędzi (u, v) w stopniach geograficznych, a następnie przeliczeniu tej wartości na mile morskie (dla zgodności jednostek).

3.3.4 Obliczenie wag

Po pobraniu danych pogodowych z API, przeliczeniu jednostek i uzgodnieniu kątów następuje odczytanie z wykresu biegunowej chwilowej prędkości jachtu. Kiedy znana jest również długość odcinka trasy możliwe jest obliczenie wagi krawędzi - czasu, w jakim jacht przepłynie ten odcinek korzystając ze wzoru na czas: $t = \frac{s}{v}$, gdzie t oznacza czas, s drogę - długość odcinka trasy, v - chwilową prędkość jachtu przy danych parametrach pogody. Otrzymany czas w godzinach jest traktowany jako waga krawędzi i nakładany na graf.

3.4 Zastosowane algorytmy

Po skonstruowaniu grafu i naniesieniu na niego wag możliwe jest przejście do problemu stricte - obliczenia czasu, w jakim jacht pokona całą zadaną trasę. Sprowadza się to rozwiązania problemu najkrótszej ścieżki między dwoma wierzchołkami.

3.4.1 Problem najkrótszej ścieżki

Dla danego grafu ważonego $G = (V, E, w)$, gdzie V oznacza zbiór wszystkich wierzchołków grafu, E zbiór krawędzi, w jest funkcją wagową $w : E \to \mathbb{R}$ powyższy problem ma różne warianty:

(a) Problem najkrótszej ścieżki między parą wierzchołków polega na znalezieniu w zadanym grafie ścieżki między zadanymi wierzchołkami u i $v \in V$ o najmniejszej wadze. Przez ścieżkę w grafie należy rozumieć taki ciąg wierzchołków, że dla każdych dwóch kolejnych wierzchołków w tym ciągu istnieje krawędź je łącząca, wraz z tymi krawędziami, bez powtórzonych krawędzi. Waga ścieżki to suma wag krawędzi na danej ścieżce.

(b) Problem najkrótszych ścieżek z jednym źródłem polega na znalezieniu najkrótszych ścieżek z wyróżnionego wierzchołka $s \in V$ nazywanego źródłem do każdego wierzchołka $v \in V$.

Problem najkrótszej ścieżki ma rozwiązanie, kiedy w grafie istnieje ścieżka, a więc kiedy graf jest spójny. Oprócz tego ze źródła nie może być osiągalny żaden cykl o ujemnej wadze. Wtedy zachodzi tzw. problem braku najkrótszej ścieżki.

Problem opisywany w tej pracy sprowadza się do problemu najkrótszej ścieżki między dwoma wierzchołkami. Należy jednak wziąć pod uwagę, że jeśli dla danego grafu zostanie rozwiązany problem najkrótszych ścieżek z jednym źródłem, to zostanie również rozwiązany problem najkrótszej ścieżki między parą wierzchołków. Dodatkowo dla problemu najkrótszej ścieżki między parą wierzchołków nie jest znany żaden algorytm, który w pesymistycznym przypadku rozwiąże go szybciej niż najlepszy znany algorytm dla problemu z jednym źródłem (patrz [1, rozdział 24]).

Z powyższych powodów w tej pracy wykorzystane zostały algorytmy znajdowania najkrótszej ścieżki z jednym źródłem. Jednak zaistniały problem dzieli się na dwa, różne podproblemy, wymagające odmiennych rozwiązań.

3.4.2 Podproblem I

Podproblem I występuje wtedy, gdy znane są wszystkie wagi w grafie. Dzieje się tak, kiedy zakres danych pobranych z prognozy pogody obejmuje cały planowany rejs. Wtedy rozwiązaniem jest modyfikacja klasycznego algorytmu Dijkstry, w którym wagi zmieniają się w czasie. Zmiana wag w czasie polega na tym, że wagi są obliczane na podstawie prognozy pogody dla danej godziny. Konieczne jest uwzględnienie tego, w którym momencie jacht dopłynął do punktu reprezentowanego przez dany wierzchołek i obliczenie wag krawędzi sąsiadujących od tej godziny.

Klasyczny algorytm Dijkstry rozwiązuje problem najkrótszej ścieżki z jednym źródłem dla danego grafu $G = (V, E, w)$. $w(u,v)$ to waga krawędzi $(u, v) \in E$. W algorytmie Dijkstry wagi wszystkich krawędzi muszą być nieujemne. Rozpatrywany w pracy problem to zapewnia - waga krawędzi reprezentuje czas, w jakim jacht pokona dany odcinek trasy, jest więc zawsze nieujemna.

W algorytmie wykorzystywane są:

- s - wierzchołek źródłowy
- $d[v]$ - oszacowanie wagi najkrótszej ścieżki - górne ograniczenie wagi najkrótszej ścieżki z s do v
- $\pi[v]$ - poprzednik wierzchołka v, który jest albo innym wierzchołkiem, albo jest równy NIL
- Q - kolejka priorytetowa z operacją `extractMin`; operacja `extractMin(`Q`)` usuwa i zwraca element ze zbioru Q o najniższym priorytecie
- $Adj[v]$ - lista sąsiedztwa zawierająca wszystkie wierzchołki będące sąsiadami wierzchołka v w grafie G

Pseudokod 3.2: Zmodyfikowany algorytm Dijkstry

```
   Input: graf G = (V, E, w)
   Output: tablica d z wagami najkrótszych ścieżek
1  for każdy wierzchołek v ∈ V do
2      d[v] ← ∞
3      π[v] ← NIL
4  d[s] ← 0
5  Q ← V
6  while Q ≠ ∅ do
7      u ← extractMin(Q)
8      for każdy wierzchołek v ∈ Adj[u] do
9          w(u,v) ← calculateWeight(d[u])
10         if d[v] > d[u] + w(u,v) then
11             d[v] ← d[u] + w(u,v)
12             π[v] ← u
```

W wierszach 1-4 inicjowane są wartości początkowe atrybutów d i π. W wierszu 5 tworzona jest kolejka priorytetowa Q ze wszystkich wierzchołków grafu G, gdzie priorytetem jest wielkość d. Następnie dopóki w kolejce Q są wierzchołki: do zmiennej u jest przypisywany i jednocześnie usuwany z kolejki Q wierzchołek o najniższym priorytecie. W wierszu 9 widoczna jest modyfikacja algorytmu. Polega ona na wykonaniu funkcji `calculateWeight(`$d[u]$`)`, która oblicza wagę krawędzi (u, v) na podstawie wielkości wagi najkrótszej ścieżki od s do u, czyli czasu, który minął od wypłynięcie do dotarcia do u. Na podstawie tego czasu do obliczenia wagi krawędzi (w sposób opisany w 3.4) jest użyta odpowiednia godzina. Później następuje relaksacja wszystkich krawędzi wychodzących z wierzchołka u. Metoda relaksacji (osłabienia ograniczeń) krawędzi (u,v) polega na sprawdzeniu, czy przechodząc przez wierzchołek u, można znaleźć krótszą od dotychczas najkrótszej ścieżki do v i, jeśli istnieje taka możliwość, aktualizowane są atrybuty $d[v]$ i $\pi[v]$.

Złożoność

Wprowadzona modyfikacja algorytmu nie zmienia jego złożoności obliczeniowej w stosunku do klasycznego algorytmu Dijsktry, ponieważ złożoność dodanej funkcji `calculateWeight` wynosi $O(1)$. Algorytm opisany

w pseudokodzie 3.2 wykorzystuje kolejkę priorytetowa Q, na której wykonywane są 3 operacje: `extractMin` (wiersz 7), `insert` - przy tworzeniu kolejki (wiersz 5) i `decreaseKey` - zmniejszanie wartości klucza (priorytetu) używana w procedurze relaksacji (wiersz 11). Operacje `extractMin` i `insert` są wykonywane po razie dla każdego wierzchołka, a operacja `decreaseKey` co najwyżej $|E|$ razy. Czas działania algorytmu zależy od sposobu implementacji kolejki priorytetowej. W tej pracy została wykorzystana implementacja tablicowa, dla której czas operacji `extractMin` wynosi $O(|V|)$, a czas operacji `insert`/`decreaseKey` wynosi $O(1)$. Łączny czas działania zmodyfikowanego algorytmu Dijkstry wynosi więc:

$$O(|V| \cdot \texttt{extractMin} + |V| \cdot \texttt{insert} + |E| \cdot \texttt{decreaseKey}) = O(|V|^2 + |V| + |E|) = O(|V|^2)$$

Możliwe jest przyspieszenie działania algorytmu Dijkstry poprzez zaimplementowanie kolejki priorytetowej za pomocą kopca d-arnego albo kopca Fibonacciego.

Rozwiązanie to nie zostało wykorzystane w tej pracy, ponieważ w praktyce najwięcej czasu w działaniu programu zajmują niezbędne zapytania do API pogodowego i to ich przyspieszenie było najważniejsze. Dodatkowo ze względu na to, że używany w pracy graf jest grafem rzadkim należałoby wykorzystać kopiec d-arny. Optymalne jest użycie takiego kopca d-arnego, w którym $d \approx |E|/|V|$ [2]. Pierwszym planowanym rozszerzeniem pracy (o czym zostanie jeszcze wspomniane) jest jednak zmiana stosunku liczby krawędzi do liczby wierzchołków grafu. Ponadto implementacja kolejki za pomocą kopca jest szybsza od implementacji tablicowej dopiero powyżej pewnej liczby wierzchołków w grafie, np. algorytm używający kolejki zaimplementowanej za pomocą kopca binarnego jest wydajniejszy, gdy $|E|$ jest mniejsze od $|V|^2/log_2(|V|)$. Ponieważ dla grafu użytego w pracy zachodzi $|E| = 12|V|$, powyższa nierówność jest spełniona dla $|V| > 74$. W praktyce wiele wykonań programu jest obsługiwane dla grafów, w których liczba wierzchołków jest mniejsza.

3.4.3 Podproblem II

Podproblem II ma miejsce, kiedy nie są znane wszystkie wagi krawędzi w grafie, a jedynie ich część - pozostałe są danymi niepewnymi. Dzieje się tak, gdy szacowany czas rejsu jest dłuższy niż zakres dostępnych danych pogodowych - kiedy rejs jest długi albo kiedy jest planowany z większym wyprzedzeniem.

W takim przypadku zastosowano następujące rozwiązanie: do momentu, w którym dostępne są dane pogodowe używane jest rozwiązanie z podproblemu I - wagi krawędzi są obliczane na podstawie szacowanego czasu dopłynięcia do danego punktu. Jeśli szacowany czas dopłynięcia przekracza zakres dostępnych danych, to waga krawędzi jest liczona na podstawie ostatnich dostępnych danych pogodowych dla danego wierzchołka.

Podczas tworzenia tej pracy jednym z rozważanych rozwiązań podproblemu II było zastosowanie koncepcji minmax regret (dokładny opis [3]). Koncepcja ta opiera się na minimalizacji maksymalnego żalu - podejmowana decyzja powinna być dobra we wszystkich scenariuszach. Schemat jej działania dla problemu poruszanego w pracy można przedstawić następująco:

1. Tworzone są przedziały $\langle min, max \rangle$, gdzie min i max oznaczają odpowiednio minimalny i maksymalny czas, w którym jacht mógłby przebyć dany odcinek trasy.
2. Jako wagi w grafie brane są środki wyznaczonych przedziałów.
3. Stosowany jest klasyczny algorytm Dijkstry, który wyznacza ścieżkę aproksymacyjną.
4. Otrzymywany jest koszt pesymistyczny.

Zastosowanie koncepcji minmax regret najprawdopodobniej sprawdziłoby się w tej pracy, ponieważ żeglarstwo jest jedną z tych dziedzin, w której nie należy przyjmować wariantu optymistycznego, a pesymistyczny. Jednak pomimo rozwiązania problemu modelowania danych niepewnych generowałoby to inny problem - nie są bowiem znane krańce przedziałów. Ich wyznaczenie wymagałoby dostępu do wielu różnych danych, parametrów pogody itp., dodatkowo pozostałaby kwestia, na podstawie jakiego przedziału czasowego należy je obliczać: tygodniowego, miesięcznego czy może rocznego. Nie wiadomo też, na ile takie wyznaczenie wartości min i max miałoby odzwierciedlenie w rzeczywistości. Z tego powodu koncepcja minmax regret nie została użyta w tej pracy, jest jednak jednym z pomysłów na jej rozszerzenie w przyszłości, o ile będą dostępne niezbędne dane.

Analiza projektowa

4.1 Wymagania funkcjonalne

- Możliwość wygodnego wprowadzenia danych o porcie startowym i końcowym, a także modelu jachtu poprzez ich wybór z listy rozwijanej
- Możliwość wprowadzenia daty za pomocą kalendarza z zaznaczonymi dostępnymi datami (np. daty przeszłe są nieaktywne)
- Prezentowanie znalezionej optymalnej trasy i obliczonego czasu
- Prosta struktura plików zawierających dane z wykresów biegunowych prędkości, aby umożliwić łatwe dodawanie danych z własnych jachtów

4.2 Wymagania niefunkcjonalne

- Możliwość uruchomienia aplikacji pod systemem operacyjnym Windows 10
- Prosta instalacja
- Obliczenie szukanych wartości z możliwie dużą dokładnością i w możliwie krótkim czasie
- Otwartość aplikacji na rozbudowę

4.3 Przypadki użycia

Nazwa: Planowanie czasu rejsu
Aktor: Użytkownik
Cel: Obliczenie czasu przepłynięcia określonej trasy
Warunki wstępne: Dostęp do Internetu
Główny scenariusz:

1. Użytkownik wybiera z listy port, z którego zamierza rozpocząć rejs
2. Użytkownik wybiera z listy port, w którym zamierza zakończyć rejs
3. Użytkownik wybiera z listy model jachtu, jakim zamierza się poruszać
4. Użytkownik wybiera z kalendarza dzień, od którego zamierza rozpocząć rejs
5. Program na podstawie wprowadzonych danych wyznacza optymalną trasę i oblicza czas jej pokonania
6. Program prezentuje znalezioną trasę i czas na ekranie urządzenia

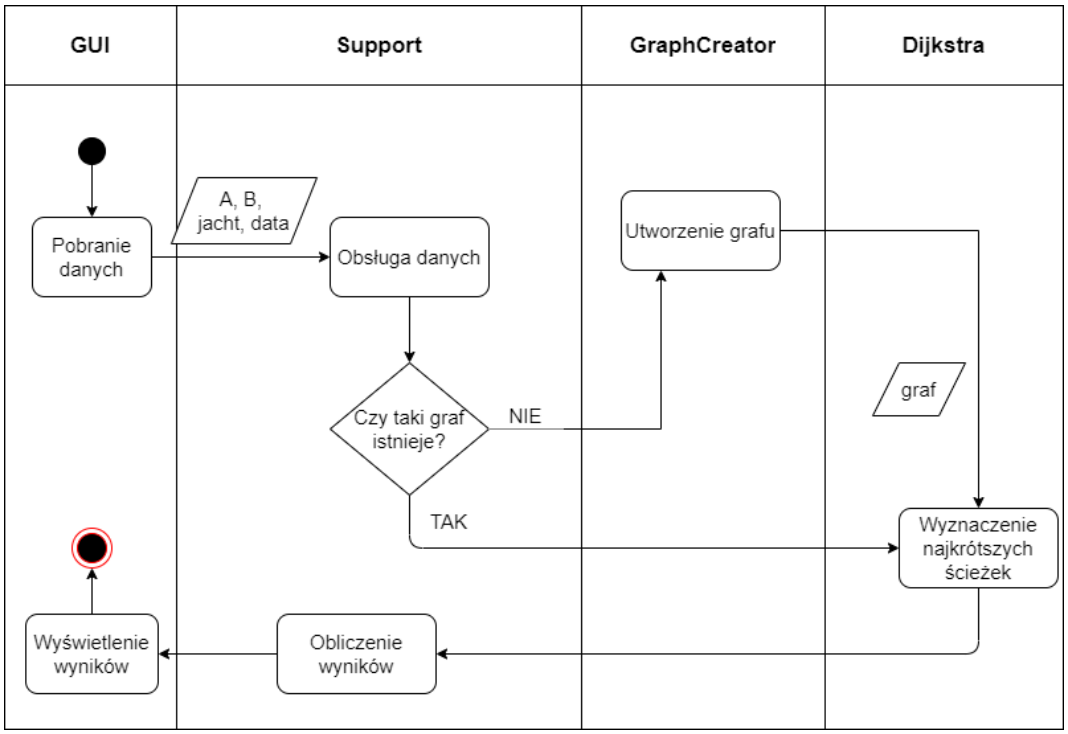
Alternatywne scenariusze:

- Komunikat o niekorzystnych warunkach pogodowych dla danej trasy

- Komunikat o braku dostępu do Internetu
- Komunikat o wykorzystaniu dostępnych zapytań API

4.4 Diagram swimlane

Do modelowania architektury systemu, w szczególności procesów zachodzących w programie wykorzystany został diagram swimline. Diagram 4.1 przedstawia w sposób uproszony proces od wprowadzenia prze użytkownika parametrów do wyświetlenia wyników końcowych. Aktywności zostały pogrupowane według odpowiedzialnych za nie klas. Przedstawiony został przepływ zarówno aktywności, jak i obiektów.



Rysunek 4.1: Diagram swimline obrazujący przepływ operacji w programie

Implementacja systemu

5.1 Opis technologii

Java

Do implementacji aplikacji został wykorzystany język Java w wersji 15 (pełna dokumentacja: [6]). Wybór języka Java wynikał z dostępności wielu gotowych bibliotek oraz rozwiązań pozwalających na integrację z usługami zewnętrznymi. Przykłady bibliotek użytych w pracy to HTTP Client API do obsługi zapytań do API zewnętrznego oraz biblioteka graficzna Swing do stworzenia GUI. Dodatkowo kod napisany w Javie kompiluje się do kodu pośredniego wykonywanego na maszynie wirtualnej Javy, jest więc niezależny od systemu operacyjnego. To sprawia, że aplikacja będzie dostępna dla szerszego grona użytkowników. Łatwiejsza będzie też ewentualna migracja na aplikację mobilną dla systemu Android.

Maven

Projekt został zbudowany przy użyciu Apache Maven - narzędzia, które automatyzuje budowę oprogramowania napisanego w jezyku Java. [7]

GSON

GSON jest biblioteką języka Java typu open-source. Umożliwia ona przekształcenie danych w formacie JSON do objektów języka Java (ang. *Java Object*) i odwrotnie. W tej pracy biblioteka GSON została użyta do konwertowania danych pobieranych z API pogodowego.

JUnit4

JUnit4 [8] jest biblioteką służącą do pisania testów jednostkowych dla programów napisanych w języku Java. Wszystkie testy jednostkowe dla aplikacji zostały napisane przy jej użyciu.

Storm Glass API

Storm Glass API [9] dostarcza dane meteorologiczne dla wszystkich mórz i oceanów na świecie. Jego atutem jest duża dokładność, wysoka rozdzielczość i duży zakres czasowy danych, a także częsta aktualizacja prognozy pogody oraz wiele różnych parametrów. W tej pracy ze Storm Glass API są pobierane dane o prędkości i kierunku wiatru dla danego punktu oraz informacja, czy dany punkt jest morzem czy stałym lądem.

5.2 Omówienie kodów źródłowych

Kompletne kody źródłowe znajdują się na płycie CD dołączonej do pracy.

Kod źródłowy 5.1 przedstawia fragment kodu odpowiedzialnego za asynchroniczne fetchowanie danych z API z użyciem Java HTTP Client API. Wcześniej dla każdego wierzchołka grafu tworzony jest odpowiedni endpoint (zależny od współrzędnych punktu i potrzebnych parametrów pogodowych) - metoda `createTargetsMap`.

Kod źródłowy 5.1: Asynchroniczne fetchowanie danych z API

```
private final ExecutorService executorService = Executors.newFixedThreadPool(5);

private final HttpClient httpClient = HttpClient.newBuilder()
        .executor(executorService)
        .version(HttpClient.Version.HTTP_2)
        .connectTimeout(Duration.ofSeconds(120))
        .build();

public Map<Vertex, CompletableFuture<String>> getAllResponses(Set<Vertex> vertices) {
    Map<Vertex, URI> targets = createTargetsMap(vertices);
    return targets.entrySet()
        .stream()
        .map(e -> new AbstractMap.SimpleEntry<>(e.getKey(), httpClient
            .sendAsync(
                HttpRequest.newBuilder(e.getValue()).header(authorization,key).GET().buid(),
                HttpResponse.BodyHandlers.ofString())
            .thenApply(HttpResponse::body)))
        .collect(Collectors.toMap(
            Map.Entry::getKey,
            Map.Entry::getValue
        ));
}
```

Kod źródłowy 5.2 przedstawia obliczenie kąta między geograficzną północą (0°) a krawędzią grafu między dwoma wierzchołkami z dokładnością do 30° z użyciem wbudowanej w Javę klasy `Math`. W programie metoda ta jest używana przy dołączaniu wierzchołka reprezentującego port końcowy do stworzonego już grafu. Jest ona dokładnie opisana w podrozdziale naniesienie punktu końcowego.

Kod źródłowy 5.2: Obliczanie kąta między północą a krawędzią grafu

```
public int calculateEdgeAlpha(Vertex p, Vertex q) {
    double alpha;
    double a = (q.getY() - p.getY()) / (q.getX() - p.getX());
    double phi = Math.toDegrees(Math.atan(a));
    if (p.getX() <= q.getX()) {      // kursy 0-180
        alpha = 90.0 - phi;
    } else {                         // kursy 180+
        alpha = 180.0 + 90.0 - phi;
    }
    return (int) Math.round(alpha / 30) * 30;
}
```

Instalacja i wdrożenie

6.1 Instalacja

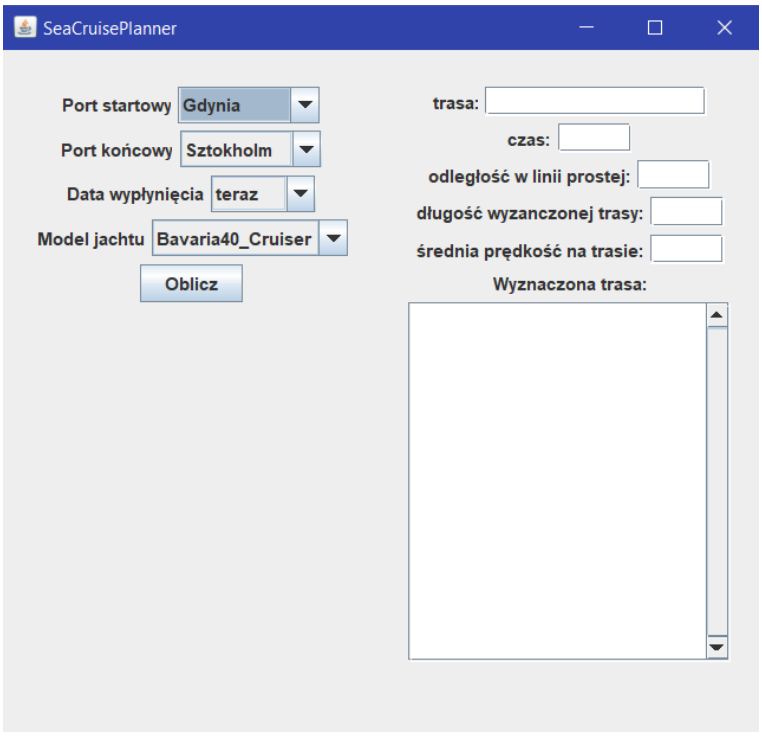
Aby zainstalować aplikację niezbędne są: Windows 10, Java JDK 15 oraz Maven. W celu uruchomienia aplikacji z linii poleceń należy z folderu zawierającego kod projektu (`SeaCruisePlanner`) wykonać następujące komendy:

```
mvn compile
mvn install
mvn package
mvn -Dmaven.test.skip=true package
mvn exec:java -Dexec.mainClass=SeaCruisePlanner
```

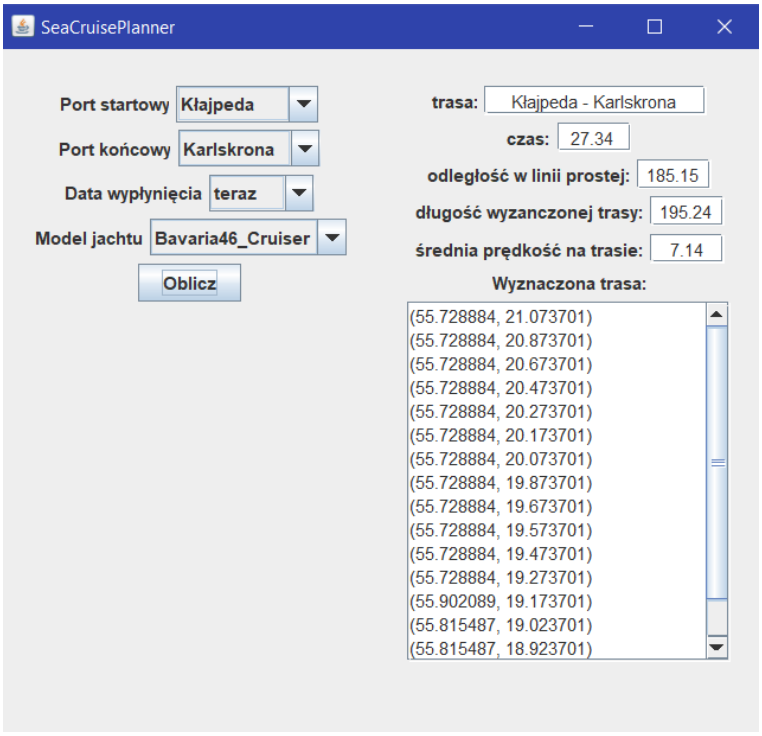
6.2 Instrukcja i przedstawienie aplikacji

Po uruchomieniu programu użytkownik ma przed sobą okienko aplikacji jak na rysunku 6.1. Po lewej stronie należy wprowadzić, korzystając z list rozwijanych następujące parametry: port startowy, port końcowy, planowaną datę rozpoczęcia rejsu oraz model jachtu. Po wybraniu parametrów należy kliknąć przycisk ”Oblicz”. Następnie należy odczekać chwilę, która (zwłaszcza dla dłuższych tras) może trwać kilkadziesiąt sekund, ze względu na konieczność skonstruowania grafu, pobrania danych pogodowych i dokonania obliczeń.

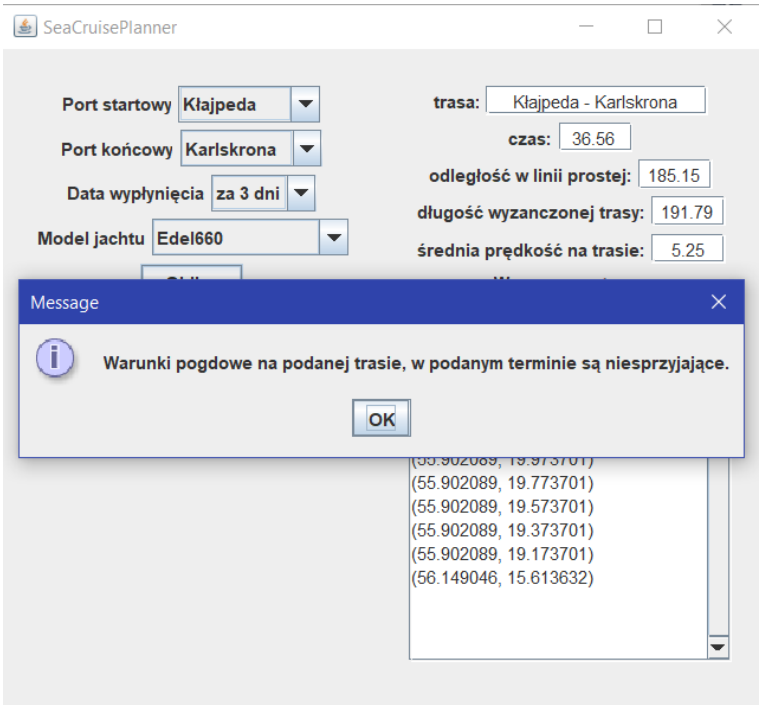
Następnie obliczone wartości są wyświetlane po prawej stronie, jak na rysunku 6.2. Są to dane o porcie startowym i końcowym, obliczony czas, długość trasy w linii prostej oraz obliczona przez aplikację, średnia prędkość jachtu na trasie oraz lista współrzędnych punktów, przez które prowadzi wyznaczona trasa. W przypadku, gdy przy kolejnym zapytaniu port startowy i końcowy nie zmienią się aplikacja korzysta ze stworzonego wcześniej grafu. W przypadku, gdy obliczenie czasu nie jest możliwe ze względu na niekorzystne warunki pogodowe (dokładniej czas przebycia trasy dąży do nieskończoności, przypadek opisany w podrozdziale 7.1) wyświetlany jest stosowny komunikat, rysunek 6.3. Chociaż może się wydawać, że aplikacja nie spełniła w takim przypadku swojej funkcji jest przeciwnie - z punktu widzenia użytkownika planującego rejs, chcącego jak najlepiej wykorzystać warunki pogodowe jest ważna informacja - może on pomyśleć o wybraniu innej trasy albo nastawić się na potrzebę używania silnika. W przypadku braku połączenia z Internetem albo wykorzystaniu dziennego limitu zapytań do API pogodowego również jest wyświetlana odpowiednia informacja.



Rysunek 6.1: Aplikacja po uruchomieniu



Rysunek 6.2: Aplikacja po dokonaniu obliczeń



Rysunek 6.3: Informacja o niekorzystnych warunkach pogodowych an trasie

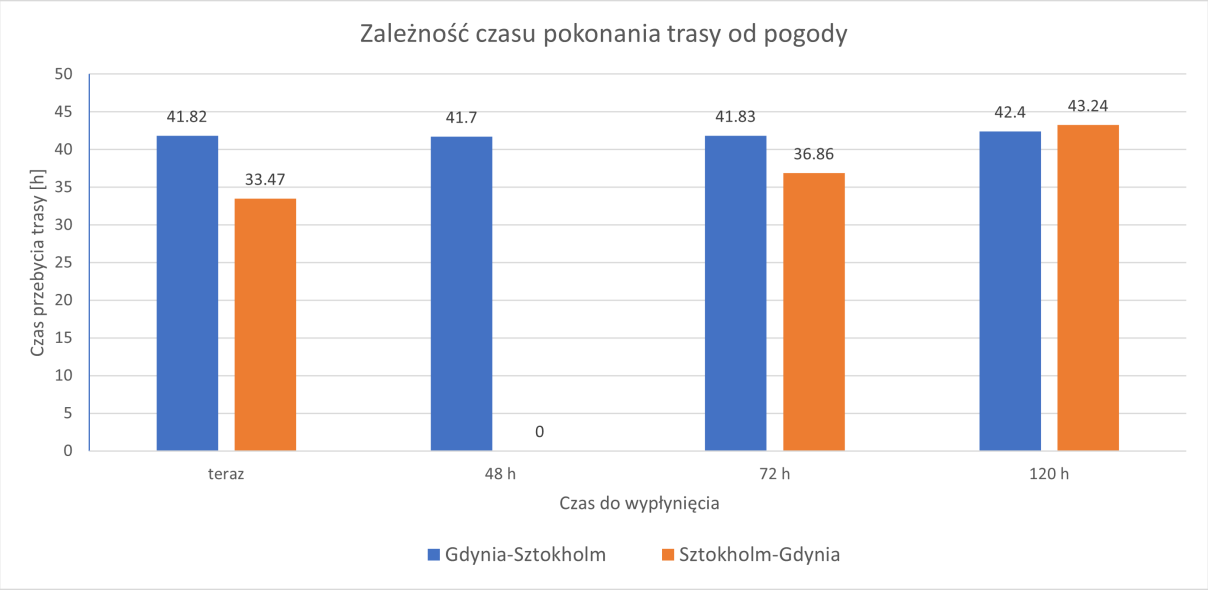
Testy

7.1 Czas przepłynięcia trasy a zmiany prognozy pogody

Jedną z funkcji aplikacji jest pokazanie zależności między zmianami parametrów pogody, a obliczonym czasem przepłynięcia przez jacht danej trasy. Różnice te dotyczą zmiany prognozy pogody w czasie, ale też zmiany trasy jachtu, a więc jego kursu względem wiatru.

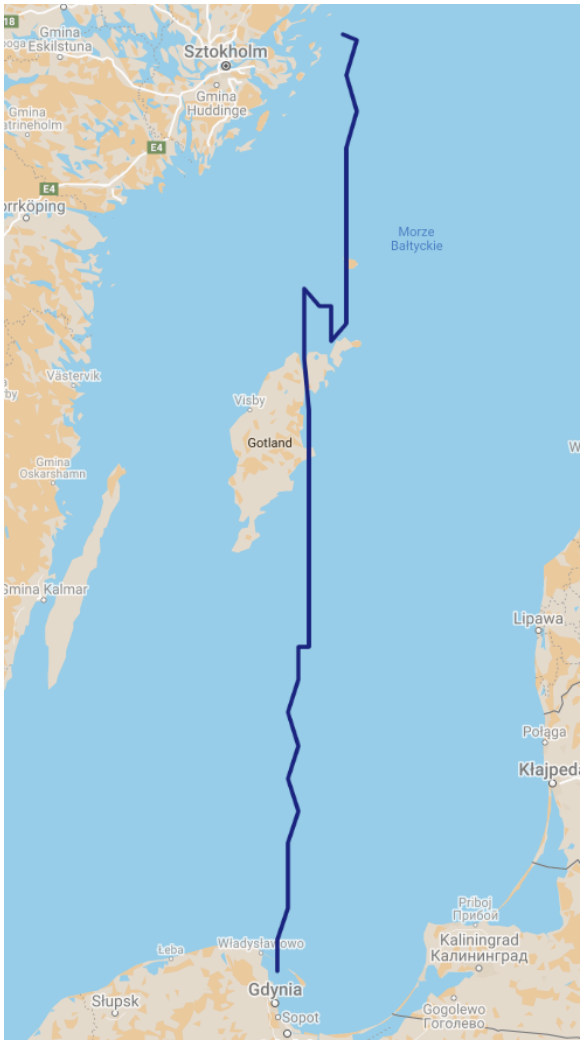
W tabeli 7.1 znajdują się wyniki testów dla trasy Gdynia-Sztokholm i Sztokholm-Gdynia (model jachtu: Bavaria 46 Cruiser, data przeprowadzenia testu: 6.1.2021). Przedstawiony został obliczony czas przebycia trasy oraz jej długość w zależności od czasu wypłynięcia: uwzględniono wypłynięcie od razu, za 2, 3 oraz za 5 dni. Porównanie otrzymanych wyników obrazuje wykres 7.1.

	Gdynia-Sztokholm *czas [h]*	Sztokholm-Gdynia *czas [h]*	Gdynia-Sztokholm *droga [NM]*	Sztokholm-Gdynia *droga [NM]*
teraz	41.82	33.47	354.34	301.8
48 h	41.7	-	309.73	-
72 h	41.83	36.86	307.67	303.58
120 h	42.4	43.24	315.18	313.03

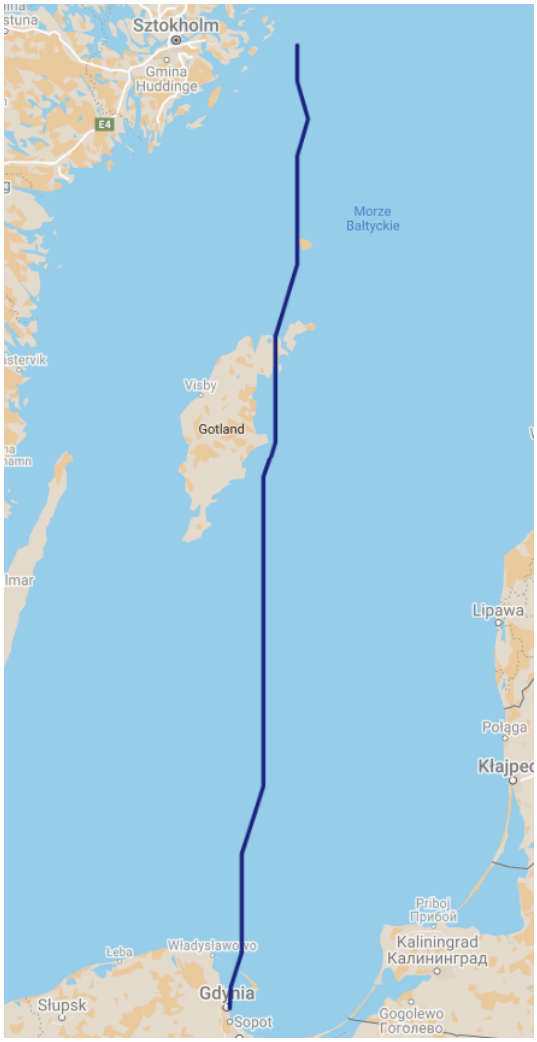


Rysunek 7.1: Zależność czasu przebycia trasy od pogody

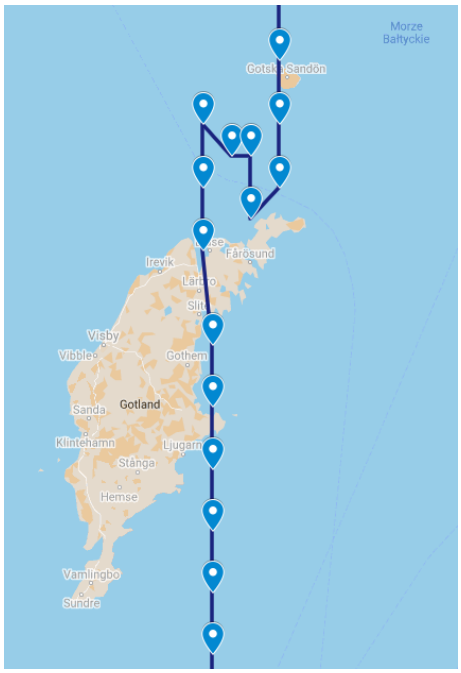
Można zauważyć na podanym przykładzie różne powiązania. Czasy pokonania tras z miasta A do miasta B i w odwrotną stronę czasami są do siebie zbliżone, np. dane dla wypłynięcia 120 h. Czasami jednak znacznie się różnią, np. dane dla wypłynięcia "teraz", gdzie różnica między czasem dla trasy Gdynia-Sztokholm a Sztokholm-Gdynia to aż 20%. Wynika to z różnic wpływu działania wiatru o określonym kierunku na jacht, W pierwszym opisanym przykładzie był to najprawdopodobniej wiatr wiejący pod kątem około 90°, który wpływał podobnie an jacht na obu trasach. W drugim opisanym przypadku był to najprawdopodobniej wiatr wiejący z północy na południe, a więc dla trasy Sztokholm-Gdynia bardziej sprzyjający niż dla trasy odwrotnej.
Różnią się też długości wyznaczonych tras. Największą różnicę widać w wynikach dla wypłynięcia "teraz", gdzie różnica ta wynosi ponad 50 mil morskich. Obie te trasy zostały narysowane za pomocą *Google My Maps* według połączenia współrzędnych wyznaczonych przez aplikację i zamieszone dla rysunkach 7.2a i 7.2b. Widać wyraźnie, że trasa Gdynia-Sztokholm była dłuższa.
Może się wydawać, że wyznaczona trasa prowadzi przez ląd. Wynika to stąd, że przy konstruowaniu grafu następuje sprawdzenie, czy dany wierzchołek jest wodą czy lądem, ale wierzchołki te są następnie łączone krawędzią bez sprawdzania, przez jaką formę terenu prowadzi ta krawędź. Widać to na rysunku 7.3.



(a) Trasa Gdynia-Sztokholm



(b) Trasa Sztokholm-Gdynia



Rysunek 7.3: Fragment trasy z zaznaczonymi krawędziami grafu

Brak wyniku

Kolejną rzeczą, na którą należy zwrócić uwagę jest brak danych dla trasy Sztokholm-Gdynia dla wypłynięcia ”za 48 h”. Oznacza to, że dla wprowadzonych danych nie było możliwości wyznaczenia trasy. Nie jest to jednak błąd. Wynika to stąd, że jacht nie może poruszać dowolnym kursem względem wiatru. Każdy jacht ma tak zwany *kąt martwy* - kąt pod którym nie może płynąć - jego prędkość jest zerowa. Dotyczy on kursów na wiatr (wiatrów wiejących od dziobu jachtu). Dla każdego modelu jachtu kąt ten jest inny, zależy bowiem od konstrukcji, najczęściej jednak obejmuje kursy od 45°. Ponieważ dokładność reprezentacji za pomocą grafu kursów jachtu względem wiatru wynosi 30°, zatem jacht z rzeczywistym kątem martwym od 45° z lewej burty do 45° z prawej burty będzie przez program postrzegany jako niezdolny do płynięcia kursami od 60° do 60°, co wyklucza aż $\frac{1}{3}$ kursów. Z tego powodu pierwszą przewidywaną modyfikacją programu jest zwiększenie dokładności kursów względem wiatru reprezentowanych przez graf z obecnych 30° do 15°.

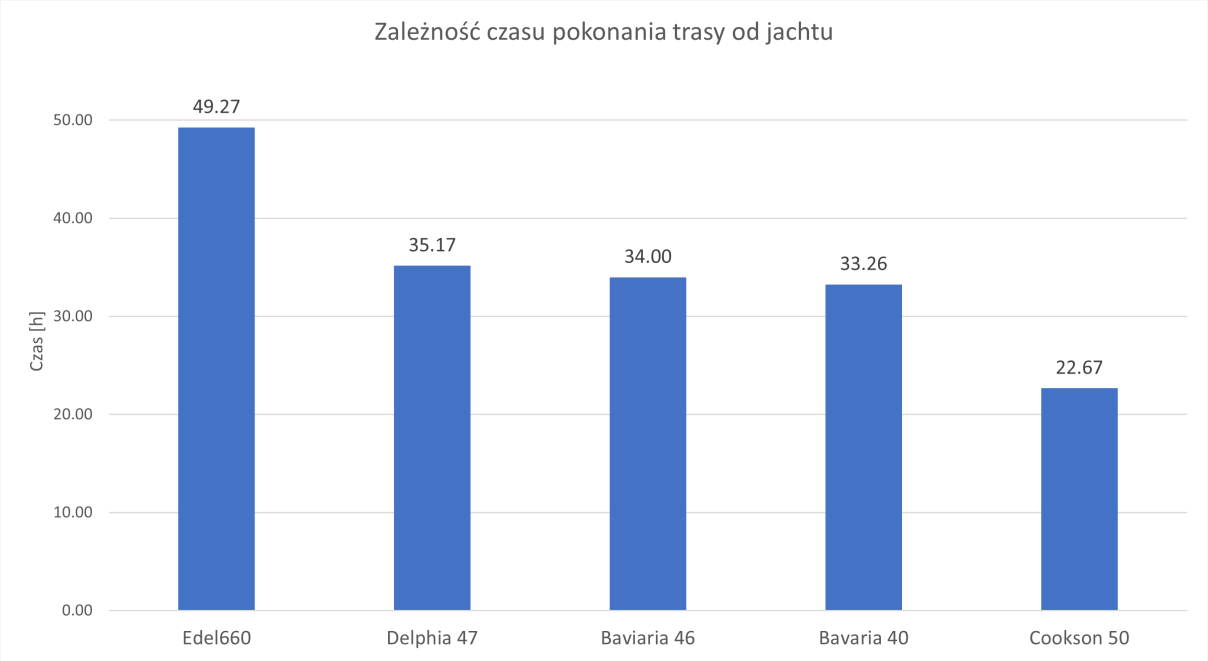
7.2 Czas przepłynięcia trasy a model jachtu

Kolejnym przeprowadzonym testem było porównanie czasu przepłynięcia określonej trasy od modelu jachtu. Użyte zostało 5 jachtów: Edel 660 (stosunkowo mały i powolny jacht), Bavaria 46 Cruiser, Bavaria 40 Cruiser, Delphia 47 (trzy jachty o podobnych parametrach) i Cookson 50 (szybki model regatowy, wariant z dodatkowym żaglem). Porównanie zostało wykonane dla trasy Świnoujście-Kłajpeda i odwrotnej. W trakcie momencie przeprowadzania testu (6.1.2021) prognoza pogody zapowiadała silny wiatr wiejący od Kłajpedy w stronę Świnoujścia. Wyniki testów przedstawia tabela 7.2. Wykres 7.4 pokazuje porównanie obliczonych czasów dla trasy Kłajpeda-Świnoujście. Wyniki dla trasy Kłajpeda-Świnoujście w zadowalający sposób pokazują różnice w i podobieństwa w charakterystyce jachtów.

Dla trasy przeciwnej, podobnie jak miało to miejsce w jednym z przypadków opisanym w teście wyżej obliczone czasy dążyły do nieskończoności. Wynika to właśnie z wiatru wiejącego do Świnoujścia, który uniemożliwiał wypłynięcie z portu i dalsze poruszanie się na samych żaglach. Wyjątkiem jest tutaj jacht Delphia 47, dla którego czas, został policzony. Przyczyną tego jest, fakt, że jacht ten ma najmniejszy kąt martwy ze wszystkich wymienionych i jako jedyny jest w stanie płynąć pod wiatr nawet kursem kątem 30°. Oczywiście

jego prędkość w takim kursie nie jest duża, co również można zaobserwować analizując wyniki.

Jacht	Kłajpeda-Świnoujście	Świnoujście-Kłajpeda
	Czas [h]	Czas [h]
Edel 660	49.27	-
Delphia 47	35.17	52.84
Baviaria 46 Cruiser	34.00	-
Bavaria 40 Cruiser	33.26	-
Cookson 50	22.67	-



Rysunek 7.4: Zależność czasu pokonania trasy od jachtu

Podsumowanie

W pracy udało się zrealizować większość założeń. Zastosowane algorytmy okazały się być dobrze dobrane do postawionego problemu, a ich implementacja dała zadowalającą szybkość ich wykonania.

Niestety pomimo zastosowania asynchronicznego fetchowania danych z API pogodowego ich pozyskiwanie trwa wciąż stosunkowo długo (jest to proces, który w największej mierze odpowiada za czas wykonywania się programu). Niestety ze względu na ograniczenia samego API (problemy z obsługą wielu zapytań asynchronicznych) większe przyspieszenie tego procesu nie było możliwe. Gdyby niniejsza praca była aplikacją komercyjną i istniałaby możliwość pokrycia kosztów dostępów do innego API zostałoby to poprawione. Dodatkowo przy braku ograniczeń finansowych możliwe byłoby użycie w pracy danych pochodzących z kilku różnych API i wyciągnięcie średniej z otrzymanych danych dla każdego punktu. To zwiększenie dokładności pobieranych danych pogodowych pozwoliłoby zwiększyć precyzję obliczeń.

Dla lepszego działania aplikacji niezbędna byłaby natomiast zmiana kształtu grafu. Zastosowana dokładność dała dostateczne wyniki. Jej zwiększenie podniosłoby dokładność obliczeń, zwłaszcza dla przypadków kursów jachtów na wiatr (kiedy kąt między osią jachtu a kierunkiem wiatru jest mały). Zmiana ta polegałaby na wspomnianym już, zwiększaniu precyzji obliczeń poprzez zwiększenie dokładności reprezentacji poszczególnych kierunków wiatru za pomocą grafu z obecnych 30° do 15°. Oczywiście konsekwencją tego byłaby zmiana kształtu grafu - z każdego wierzchołka wychodziłyby 24 krawędzie.

Możliwości rozwoju aplikacji jest więcej, na różnych płaszczyznach. Dotyczy to również dodania nowych funkcjonalności, na przykład:

- uwzględnienie możliwości płynięcia z użyciem silnika,
- zwiększenie uwzględnianego obszaru poza Morze Bałtyckie,
- dodanie innych modeli jachtów,
- dodanie rysowania wyznaczonej trasy na mapie,
- uwzględnienie dodatkowych informacji o obszarach, przez które prowadzi sugerowana, optymalna trasa takich jak: informacja o wodach terytorialnych państw, szlaki żeglowne, lokalizacje morskich elektrowni wiatrowych i innych stałych elementów, na które niezbędne jest zwrócenie uwagi przy pokonywaniu trasy w rzeczywistości.

Biorąc pod uwagę liczne trudności i ograniczenia takie jak brak aplikacji, na których można byłby się wzorować, ograniczony dostęp do API pogodowego czy niedokładność danych z wykresów biegunowych jachtów udało się zrealizować główny cel pracy jakim było obliczenia czasu, w jakim jacht morski przebędzie daną trasę, a osiągnięta precyzja obliczeń jest zadowalająca.

Bibliografia

[1] T. Cormen, C. Leiserson, R. Rivest, C. Stein. *Wprowadzenie do algorytmów.* Wydawnictwa Naukowo-Techniczne, Warszawa, 2001.

[2] S. Dasgupta, C. Papadimitriou, U. Vazirani. *Algorytmy.* Wydawnictwo Naukowe PWN SA, Warszawa, 2010.

[3] A. Kasperski, P. Zieliński. An approximation algorithm for interval data minmax regret combinatorial optimization problems. 2005.

[4] A. Kolaszewski, P. Świdwiński. *Żeglarz jachtowy i jachtowy sternik morski.* Alma-Press Sp. z o.o., 2013.

[5] Baza wykresów biegunowych prędkości jachtów. https://jieter.github.io/orc-data/site/.

[6] Java. Web pages: https://docs.oracle.com/en/java/javase/15/docs/api/index.html.

[7] Apache maven. Web pages: https://maven.apache.org/.

[8] Junit4. Web pages: https://junit.org/junit4/.

[9] Storm glass api. Web pages: https://stormglass.io/.

Zawartość płyty CD

Na załączonej do pracy płycie CD znajdują się kody źródłowe programu, jego dokumentacja oraz treść niniejszej pracy inżynierskiej.