|  |  |  |
| --- | --- | --- |
| wordml://75.png |  | wordml://76.png |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | Imię i nazwisko studenta: Artur Kąkol |  | | | |  | | --- | | Nr albumu: 143251 | | | |  | | --- | | Studia drugiego stopnia | | | |  | | --- | | Forma studiów: stacjonarne | | | |  | | --- | | Kierunek studiów: Informatyka | | | |  | | --- | | Specjalność/profil: Algorytmy i technologie internetowe | | |  |
|  |  |
| |  | | --- | | **PRACA DYPLOMOWA MAGISTERSKA** | | |
| |  | | --- | | Tytuł pracy w języku polskim: System internetowy wspierający marszrutyzację pojazdów | | Tytuł pracy w języku angielskim: Web-based system supporting Vehicle Routing Problems | |  | | |
| |  |  |  |  | | --- | --- | --- | --- | |  | |  | | --- | | Potwierdzenie przyjęcia pracy | | | |  | |  | | --- | | Opiekun pracy |   *podpis* | |  | | --- | | Kierownik Katedry/Zakładu |   *podpis* | |  | dr inż. Krzysztof Bruniecki |  | | |
| |  | | --- | | Data oddania pracy do dziekanatu: | | |

|  |  |  |  |
| --- | --- | --- | --- |
| wordml://75.png | |  | wordml://76.png |
|  | **OŚWIADCZENIE** | | |
|  | |  | | --- | | Imię i nazwisko: Artur Kąkol  Data i miejsce urodzenia: 27.05.1992, Kościerzyna  Nr albumu: 143251  Wydział: Wydział Elektroniki, Telekomunikacji i Informatyki  Kierunek: informatyka  Poziom studiów: II stopnia  Forma studiów: stacjonarne | | | |
|  | |  | | --- | | Ja, niżej podpisany(a), wyrażam zgodę/nie wyrażam zgody\* na korzystanie z mojej pracy dyplomowej zatytułowanej: System internetowy wspierający marszrutyzację pojazdów do celów naukowych lub dydaktycznych.1 | | | |
| |  |  | | --- | --- | | Gdańsk, dnia .................................. | .....................................................  *podpis studenta* | | | | |
|  | |  | | --- | | Świadomy(a) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. z 2006 r., nr 90, poz. 631) i konsekwencji dyscyplinarnych określonych w ustawie Prawo o szkolnictwie wyższym (Dz. U. z 2012 r., poz. 572 z późn. zm.),2 a także odpowiedzialności cywilno-prawnej oświadczam, że przedkładana praca dyplomowa została opracowana przeze mnie samodzielnie.  Niniejsza(y) praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem tytułu zawodowego.  Wszystkie informacje umieszczone w ww. pracy dyplomowej, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami zgodnie z art. 34 ustawy o prawie autorskim i prawach pokrewnych.  Potwierdzam zgodność niniejszej wersji pracy dyplomowej z załączoną wersją elektroniczną. | | | |
| |  |  | | --- | --- | | Gdańsk, dnia .................................. | .....................................................  *podpis studenta* | | | | |
|  | Upoważniam Politechnikę Gdańską do umieszczenia ww. pracy dyplomowej w wersji elektronicznej w otwartym, cyfrowym repozytorium instytucjonalnym Politechniki Gdańskiej oraz poddawania jej procesom weryfikacji i ochrony przed przywłaszczaniem jej autorstwa. | | |
| |  |  | | --- | --- | | Gdańsk, dnia ................................. | .....................................................  *podpis studenta* | | | | |
|  | |  | | --- | | \*) niepotrzebne skreślić | | | |
| |  | | --- | |  | | | | |
| |  |  |  |  | | --- | --- | --- | --- | | |  | | --- | | 1 | | |  | | --- | | Zarządzenie Rektora Politechniki Gdańskiej nr 34/2009 z 9 listopada 2009 r., załącznik nr 8 do instrukcji archiwalnej PG. | | | |  | | --- | | 2 | | |  | | --- | | Ustawa z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym: | | |  | |  | | --- | | Art. 214 ustęp 4. W razie podejrzenia popełnienia przez studenta czynu podlegającego na przypisaniu sobie autorstwa istotnego fragmentu lub innych elementów cudzego utworu rektor niezwłocznie poleca przeprowadzenie postępowania wyjaśniającego. | | |  | |  | | --- | | Art. 214 ustęp 6. Jeżeli w wyniku postępowania wyjaśniającego zebrany materiał potwierdza popełnienie czynu, o którym mowa w ust. 4, rektor wstrzymuje postępowanie o nadanie tytułu zawodowego do czasu wydania orzeczenia przez komisję dyscyplinarną oraz składa zawiadomienie o popełnieniu przestępstwa. | | | | | |

Streszczenie

Niniejsza praca prezentuje system internetowy wspierający marszrutyzację pojazdów. Skupia się na wariancie marszrutyzacji, w którym każdy z pojazdów ma określoną z góry pojemność, natomiast każdy z odbiorców oczekuje pewnej ilości towaru. Przedstawiony został projekt oraz opis systemu. Utworzony system został podzielony na dwie główne części. Pierwsza z nich uruchamiana jest w przeglądarce użytkownika i pozwala na edycję parametrów oraz wyświetlaniu wyników obliczeń. Druga część natomiast odpowiedzialna jest za wykonywanie obliczeń oraz zapisywaniu danych do bazy uruchomiona została na zdalnym serwerze. Dodatkowo wygenerowano oraz uruchomiono szereg testów porównujących zaimplementowane algorytmy oraz wykorzystane biblioteki.

**Słowa kluczowe:** marszrutyzacja pojazdów, system internetowy, Angular, Grails, algorytm

**Dziedzina nauki i techniki, zgodnie z wymogami OECD:** Nauki inżynieryjne i techniczne. Elektrotechnika, elektronika, inżynieria informatyczna.

Abstract

This work presents an online system supporting the routing of vehicles. It focuses on a routing variant in which each vehicle has a predetermined capacity, while each of the receivers expects a certain amount of goods. Project and the description of system have been described. Created system has been divided into two main parts. The first one is run in the user's browser and allows to edit parameters and display calculation results. The second part, however, is responsible for performing calculations and saving data to the database. In addition, a number of tests comparing the implemented algorithms and libraries were generated and launched.

**Keywords:** vehicle routing, internet system, Angular, Grails, algorithm

**OECD field of science and technology (FOS) classification:** Engineering and technology. Electrical engineering, Electronic engineering, Information engineering.

Spis treści

[Streszczenie 3](#_Toc515815356)

[Abstract 4](#_Toc515815357)

[Spis treści 5](#_Toc515815358)

[1. Wstęp 6](#_Toc515815359)

[1.1. Cele pracy 7](#_Toc515815360)

[1.2. Tematyka pracy 7](#_Toc515815361)

[1.3. Planowanie tras pojazdów 8](#_Toc515815362)

[1.4. Obliczanie odległości pomiędzy punktami 10](#_Toc515815363)

[1.5. Przegląd istniejących bibliotek 13](#_Toc515815364)

[1.6. Istniejące systemy wspierające marszrutyzację pojazdów 13](#_Toc515815365)

[1.7. Instancje testowe 14](#_Toc515815366)

[2. Projekt systemu 16](#_Toc515815367)

[2.1. Specyfikacja wymagań 16](#_Toc515815368)

[2.2. Struktura bazy danych 19](#_Toc515815369)

[2.3. Planowana architektura systemu 20](#_Toc515815370)

[2.4. Diagram procesu obliczania rozwiązania 22](#_Toc515815371)

[3. Opis systemu 24](#_Toc515815372)

[3.1. Architektura 24](#_Toc515815373)

[3.2. Część kliencka 24](#_Toc515815374)

[3.3. Część serwerowa 25](#_Toc515815375)

[3.4. Budowanie oraz instalacja aplikacji 26](#_Toc515815376)

[3.5. Opis interfejsu użytkownika 27](#_Toc515815377)

[3.6. Komunikacja 33](#_Toc515815378)

[4. Wyniki 40](#_Toc515815379)

[4.1. Środowisko testowe 40](#_Toc515815380)

[4.2. Plan testów 40](#_Toc515815381)

[4.3. Wyniki testów 43](#_Toc515815382)

[4.4. Porównanie 53](#_Toc515815383)

[5. Podsumowanie 56](#_Toc515815384)

[Wykaz literatury 57](#_Toc515815385)

[Wykaz tabel 59](#_Toc515815386)

[Wykaz rysunków 60](#_Toc515815387)

# Wstęp

We współczesnym świecie transport odgrywa bardzo ważną rolę we współczesnej gospodarce. Zarówno w usługach(transport ludzi) jak i w przemyśle(dostarczanie towarów). Dynamiczny rozwój rynku wymusza na przedsiębiorstwach utrzymania konkurencyjności. Można ją uzyskać na kilka sposobów, między innymi poprzez niższą cenę. Aktualnie przyjmuje się, że koszty paliwa w przedsiębiorstwie zajmującym się transportem drogowym mieszczą się w przedziale 15%-20% [1]. Obniżając długości tras można w dużym stopniu przyczynić się do końcowej ceny towaru.

Kolejnymi ważnym aspektem jest też, jakość realizowanych usług. Dlatego warto zadbać o to, żeby czas realizacji był jak najkrótszy oraz sprawność transportu była jak najwyższa. Pomóc w tym może oraz powinien dedykowany system internetowy, dzięki któremu w łatwy oraz szybki sposób można by było wyznaczyć trasy dla poszczególnych pojazdów.

## Cele pracy

Celem pracy jest utworzenie systemu internetowego wspomagającego marszrutyzację pojazdów. Pozwalał on będzie na łatwe zarządzanie punktami docelowymi oraz położeniem magazynu. Każdy z punktów docelowych posiada wymagania, którym należy sprostać.

Dodatkowo ocenione oraz porównane zostaną poszczególne sposoby obliczania tras dla poszczególnych pojazdów we flocie. Porównane zostaną algorytmy oraz zostanie pokazany jaki wpływ na ostateczne rozwiązanie ma sposób obliczania odległości pomiędzy poszczególnymi odbiorcami lub magazynem.

## Tematyka pracy

Niniejszy dokument skupia się na zakresie marszrutyzacji pojazdów oraz planowaniu tras dla dostaw towarów. Zagadnienia w tym zakresie mogą być bardzo skomplikowane oraz złożone. Ze względu na rodzaj różnych zastosowań oraz uwarunkowań do rozważenia można wyróżnić wiele wariantów problemu. Najważniejszymi jednak są:

* Zagadnienie komiwojażera (ang. *Travelling Salesman Problem*)
* Klasyczny problem marszrutyzacji (ang. *Vehicle Routing Problem*)

Zagadnienie komiwojażera to najprostszy wariant planowania dostaw. Uwzględnia on tylko jeden środek transportu, bez dodatkowych ograniczeń. Polega na wyznaczenie najkrótszej trasy, która odwiedza wszystkich obiorców oraz wrócić z powrotem do magazynu.

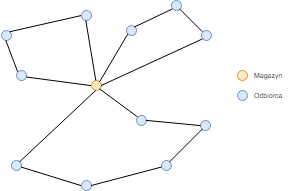
Klasyczny problem marszrutyzacji odpowiada na pytanie: „Jaki jest optymalny zestaw tras dla floty pojazdów w celu dostarczenia dóbr dla określonych odbiorców?”. Po raz pierwszy został zaprezentowany przez G.B. Dantziga oraz R. H. Ramsera w 1959 roku [2]. Jako, że zadany problem, jest rozwinięciem problemu komiwojażera to należy do problemów NP-trudnych. Istnieją liczne odmiany klasycznego zagadnienia marszrutyzacji.

Jednym z nich jest Vehicle Routing Problem with Time Windows (VRPTW). W tym rodzaju każdy z odbiorów przypisane ma ramy czasowe definiujące, kiedy dany odbiorca musi zostać obsłużony. Dopuszcza się, że pojazd może przybyć przed czasem do odbiorcy, jednak w takim wypadku musi czekać na rozładunek/załadunek.

Innymi przykładami mogą być problem dostaw uwzględniający możliwości zwrotu towaru przez klientów (ang. *Vehicle Routing Problem with Backhauls*, VRPB) lub problem, w którym istnieje wiele magazynów (ang. *Multi Depot Vehicle Routing Problem*, MDVRP)

W tej pracy główny głównie rozpatrywanym przykładem jest Capacitated Vehicle Routing Problem (CVRP), gdzie każdy z odbiorców posiada określone wymagania oraz każdy z pojazdów ma góry ustaloną pojemność. Dodatkowo wszystkie pojazdy startują oraz kończą swoją trasę w magazynie. Każdy z odbiorców może być obsłużony przez pojazd dokładnie jeden raz, co oznacza, że wymagania odbiorców nie mogą przekraczać pojemności pojazdu. Jako cel przy rozwiązywaniu tego problemy stawia się długość trasy przebytej przez pojazdy.

Przykład tego wariantu pokazany jest na Rys. 1. W tym przykładzie wszyscy odbiorcy mają wymagania równe 1, a pojazdy mają maksymalną pojemność równą 7.



Rys. 1 Przykład CVRP

Opisane wyżej warianty stanowią jednak tylko niewielką część wszystkich możliwości. Ich mnogość wynika z realnych problemów pojawiających się w przedsiębiorstwach transportowych

## Planowanie tras pojazdów

Planowanie tras pojazdów może być realizowane przez dwa różne rodzaje metod: dokładne oraz przybliżone. Wybór sposobu opiera się głównie na rozmiarze problemu, tj. na ilości odbiorców, ilości ograniczeń oraz typie floty. W przypadku problemów o mniejszym skomplikowaniu zasadnym jest używanie metod dokładnych, natomiast dla bardziej złożonych problemów ze względu na swoją złożoność obliczeniową są zdyskwalifikowane. Metody przybliżone w znacznym stopniu przyspieszają proces obliczeniowy, jednak ich rozwiązanie niekoniecznie jest optymalne (jednak może takie być).

Omawiane niżej algorytmy będą skupiały się na problemie klasy CVRP. Wiele metod zostało opracowanych do tego czasu. Przykładami dokładnych może być metoda podziału i ograniczeń (ang. branch-and-bound) lub metoda podziału i odcięć (ang. branch-and-cut) [3].

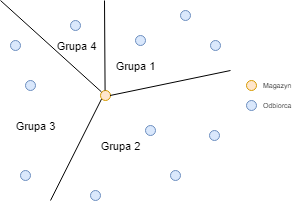
Z drugiej strony mamy jednak metody przybliżone. Jedną z najbardziej popularnych oraz najwcześniej zaproponowanych metod heurystycznych wykorzystywanych do rozwiązywania problemu klas VRP jest metoda Clarke and Wright. Główną jego ideą jest łączenie dwóch tras w jedną. Działanie algorytmu opiera się na tym, że na początku tworzymy ścieżki od magazynu do każdego z odbiorców i z powrotem *(0,i,0)* dla *(i = 1,…,n)*, oraz następnie łączymy dwie trasy *(0,…,i,0)* oraz *(0,j,…,0)* w jedną *(0,…,i,j,…,0)* generując tym samym oszczędność równą

Zostało to pokazane na Rys. 2



Rys. 2 Obliczanie oszczędności

Kolejną bardzo popularną metodą jest sweep algorithm (pierwszą jego wersję zaproponował Wren (1971) oraz Wren and Holliday (1972)). Może on zostać wykorzystany dla punktów zlokalizowanych na jakiejś płaszczyźnie euklidesowej. Najlepiej byłoby gdyby każdy z punktów (tutaj odbiorców) był przedstawiony jako para , gdzie jest to kąt oraz jest to długość promienia. Zasada jego działania jest następująca. Na początku dzielimy wszystkie punkty na grupy. Bierzemy pierwszy nieprzypisany punkt z najmniejszym kątem . I dodajemy go do grupy. Jeżeli została przekroczona ładowność pojazdu tworzymy kolejną grupę, do której przypisujemy kolejne nieprzypisane punkty. Przykładowy podział na grupy został pokazany na Rys. 3.



Rys. 3 Podział na grupy

Z tak przydzielonych grup tworzymy trasy rozwiązując problem komiwojażera. Jeżeli jest taka możliwość i wpłynie to pozytywnie na wynik, można zmieniać odbiorców pomiędzy sąsiednie trasy.

Istnieją także algorytmy metaheurystyczne, są to algorytmy ogólne wykorzystywane do rozwiązywania dowolnego problemu, który można opisać za pomocą pojęć definiowanych przez dany algorytm. Jednym z takich sposobów jest ‘Przeszukiwanie tabu’. Jest ono szeroko wykorzystywane przy rozwiązywaniu problemach marszrutyzacji. Przykłady implementacji oraz usprawnień zostały opisane w dokumencie [4]. Ogólną zasadę działania przedstawia poniższy pseudokod:

wygeneruj losowo rozwiązanie początkowe x0 ∈ X

xopt = x0

t = []

repeat

x = wybierz(N(x0),t)

t = aktualizuj\_liste\_tabu(t, x)

x0 ← x

if f(x0) > f(xopt)

then xopt ← x0

until warunek zakończenia

Gdzie początkowe rozwiązanie może być losowane na kilka sposobów, między innymi może być to całkowicie losowe rozwiązanie jak i wynik działania algorytmu oszczędnościlub innego.

## Obliczanie odległości pomiędzy punktami

Ważnym aspektem do wyznaczenia trasy dla pojazdu jest określenie odległości pomiędzy dwoma puntami na mapie. Bardzo trudno jest określić bezpośrednią odległość. Spowodowane jest to nieregularnością w powierzchni ziemi. Pomijając ten fakt jesteśmy w stanie w przybliżony sposób określić odległość.

Obliczanie możemy podzielić na dwie kategorie. Pierwsza z nich to odległość w linii prostej pomiędzy dwoma punktami, czyli ortodroma. Na pomoc w obliczeniu tej odległości przychodzi formuła Hervesina. Ma on postać:

Gdzie:

= szerokość geograficzna

= długość geograficzna

= długość promienia ziemi

Drugą możliwością jest obliczenie odległości, jaką musi pokonać samochód poruszający się po istniejących drogach. Ta jest bardziej skomplikowana do obliczenia. Do jej obliczenia potrzebne są informacje na temat dróg a następnie na jej podstawie obliczenie najkrótszej trasy. Dane na temat dróg można pobrać dzięki projektowi OpenStreetMap.

Jest to projekt społeczności internetowej tworzący otwartą oraz darmową mapę świata. Został utworzony w 2004 roku. W 2018 roku liczba zarejestrowanych użytkowników wynosiła ponad 4.5 miliona [5]. Wszystkie dane wprowadzone przez użytkowników publikowane są na otwartej licencji Open Database License. Przechowywane dane można podzielić na cztery główne kategorie:

* Węzły, które definiują punkty z ich pozycją geograficzną.
* Linie, które są listą węzłów. Używane do opisywania linii lub zamkniętych obszarów.
* Relacje, które są grupami węzłów oraz linii wraz z ich rolami. Wykorzystywane do opisu zależności pomiędzy obiektami.
* Tagi, które składają się z par klucz=wartość i stosowane są do węzłów, dróg oraz relacji.

Kompletna baza danych wydawana jest każdego tygodnia w postaci pliku XML pod nazwą *Planet.osm*. Jest to bardzo duży plik, który na dzień 2018-04-­21 zajmował 913GB. Możliwe jest jednak pobranie skompresowanej wersji w formacie PBF (Protocol Buffer Fromat). Wszystkie dane zawarte w nim są zapisane w formacie binarnym. Jest zdecydowanie mniejszy (około 40GB) i zdecydowanie szybszy przy wczytywaniu oraz analizowaniu i powinien być używany wtedy tylko, kiedy jest to możliwe. Do dyspozycji też są możliwe pobranie częściowej mapy zawierającej tylko i wyłącznie dane z określonego terenu.

Tak uzyskane dane na temat dróg oraz punktów możemy wykorzystać do obliczenia odległości pomiędzy puntami. Możliwe jest to przy pomocy bibliotek OSRM (Open Source Routing Machine) lub GraphHopper.

GraphHopper jest otwarto źródłową biblioteką napisaną w języku Java. Może być uruchomiona na wielu platformach zarówno serwerowych jak i desktopowych. Wspiera także system operacyjny Android. Może być skonfigurowany tak, aby używał algorytmu *Dijkstry* lub *A\*.* W celu szybszego obliczania tras wykorzystuje także technikę nazwaną *contraction hierarchies.* Dzięki temu, że jest ona udostępniona na licencji Apache License pozwala na dowolną edycję oraz integrację z nawet komercyjnymi produktami.

Domyślnie biblioteka ta używa algorytmu Dijkstry. Służy on do wyznaczania najmniejszej odległości od ustalonego wierzchołka *s* do wszystkich pozostałych w grafie. Początkowo mamy zbiór *Q, który* reprezentuje wszystkie wierzchołki w grafie. Dodatkowo mamy wektor *D[i]* w którym przechowujemy odległości od wierzchołka *s* do wierzchołka *i*. Początkowo wektor ten posiada pierwszy wiersz macierzy wag *A.* Algorytm ten można przedstawić przy pomocy następujących kroków:

1. Ze zbioru pobierz wierzchołek *v* o najmniejszej wartości oraz usuń go ze zbioru
2. Dla każdego sąsiada *i* wierzchołka *v* sprawdź czy
3. Jeżeli tak jest to
4. Dopóki istnieje jeszcze jakiś wierzchołek w zbiorze to przejdź do kroku 1.

Przykładowo mamy graf, dla którego macierz odległości została zaprezentowana w Tabeli 1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** |
| **0** | 0 | 35 | 75 | 45 | 20 | 7 |
| **1** | 26 | 0 | 67 | 10 | 37 | 28 |
| **2** | 3 | 35 | 0 | 36 | 7 | 5 |
| **3** | 7 | 9 | 20 | 0 | 2 | 78 |
| **4** | 75 | 35 | 1 | 15 | 0 | 2 |
| **5** | 18 | 6 | 18 | 6 | 10 | 0 |

Tabela 1 Macierz odległości

Na podstawie powyższych danych zostaje obliczony wektor *D*. Jego poszczególne wartości przestawione są w Tabeli 2. Najmniejsza wartość jest pogrubiona.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Q* | *D[1]* | *D[2]* | *D[3]* | *D[4]* | *D[5]* |
| {1,2,3,4,5} | 35 | 75 | 45 | 20 | **7** |
| {1,2,3,4} | **13** | 25 | 13 | 17 |  |
| {2,3,4} |  | 25 | **13** | 17 |  |
| {2,4} |  | 25 |  | **15** |  |
| {2} |  | **16** |  |  |  |
| {} |  |  |  |  |  |

Tabela 2 Kolejne kroki algorytmu

Złożoność obliczeniowa podanego algorytmu zależna jest od tego jak będziemy przechowywać zbiór *Q*. Jeżeli oznaczymy liczbę wierzchołków przez *n*, w takim przypadku liczba krawędzi w grafie pełnym (a taką sytuację mamy w przypadku marszrutyzacji pojazdów) wynosi *n2.* Każda z krawędzi będzie analizowana dokładnie raz. Dodatkowo zbiór *Q* zostaje przeszukiwany *n* razy w poszukiwaniu najmniejszej wartości. W przypadku przechowywania go w zwykłej tablicy złożoność czasowa będzie wynosiła . W przypadku wykorzystania kopca, jako struktury danych dla zbioru *Q* zostaje ona zmniejszona do .

## Przegląd istniejących bibliotek

Problem marszrutyzacji pojazdów nie jest nowy, z tego powody istnieją już gotowe biblioteki rozwiązujące ten problem.

Pierwszą rozpatrywaną biblioteką jest Jsprit. Jest to darmowa biblioteką napisana w języku Java. Rozwiązuje szereg wariantów problemu marszrutyzacji. Bazuje na jednej meta-heurystycznej metodzie zaproponowanej w 2000 roku przez Schrimpf et al. [6]. Opiera ona się na zasadzie zburz i odbuduj. Polega ona na tym, że wybieramy pewne startowe rozwiązanie. To rozwiązanie rozstaje zrujnowane poprzez usunięcie z niego losowych odbiorców. Poprzez to usunięcie powstaje nam częściowe rozwiązanie. Do tego częściowego rozwiązania następnie dodawane są usunięci wcześniej odbiorcy. Ten etap nazywa się odbudowywaniem. Jeżeli otrzymane rozwiązanie jest lepsze od poprzedniego wybieramy je jako aktualne i dalsze obliczenia wykonujemy. Tak utworzone rozwiązanie poddawane jest znów temu samemu procesowi dopóki kryterium końca nie zostanie spełnione (ilość iteracji, czas wykonywania). Twórcy tej biblioteki przekonują, że jest to jedno z lepszych podejść do rozwiązywania skomplikowanych problemów z wieloma ograniczeniami. Dodatkowo z uwagi na jego elastyczność może zostać użyty w wielu wariantach marszrutyzacji pojazdami.

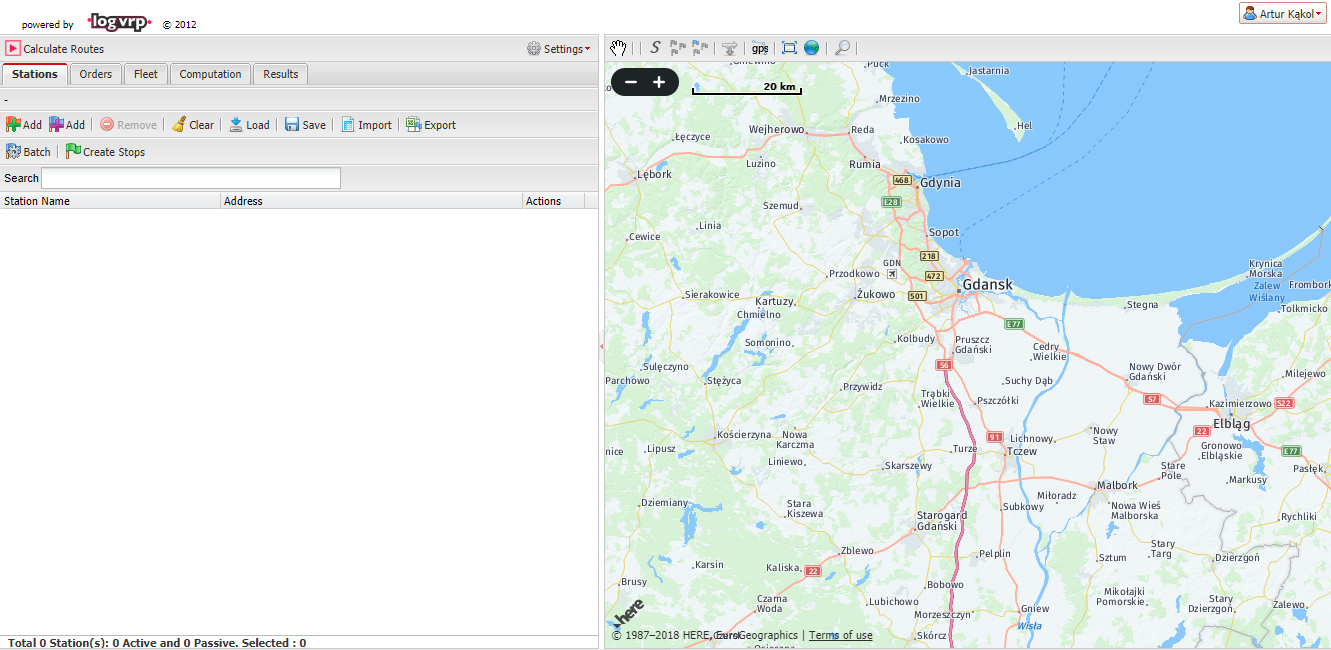
Kolejną biblioteką jest Google Optimization Tools. Jest napisana oraz rozwijana przez firmę Google. Została napisana w języku C++. Jednak możliwe jest jej użycie w także innych językach (Python, C# oraz Java). Nie specjalizuje się ona głównie na rozwiązywaniu problemu marszrutyzacji pojazdów, jednak jest to jedna z jej możliwości. W bibliotece zostało zaimplementowane wiele algorytmów. Początkowo system generuje początkowe rozwiązanie, dostępnych jest do wyboru 13, między innymi opisywany wyżej C&W Savings lub Sweep. Kolejno początkowo wygenerowane rozwiązanie jest przekazywane do poprawienia wyniku przez jedną z zaimplementowanych metod metaheurystycznych. Do wyboru jest 5 różnych metod, między innymi Symulowane Wyżarzanie lub Przeszukiwanie Tabu.

OptaPlanner jest ostatnią z rozpatrywanych bibliotek. Napisano ją w Javie. Zajmuje się między innymi marszrutyzacją pojazdów, w tym celu wykorzystuje różnego rodzaju metody rozwiązywania. Jest sponsorowana przez firmę Red Hat.

## Istniejące systemy wspierające marszrutyzację pojazdów

Idąc krok dalej, istnieją też już gotowe systemy rozwiązujące problem marszrutyzacji. Większość z nich jest rozwiązaniami o zamkniętym kodzie źródłowym oraz udostępniane za opłatą, lub za darmo z bardzo ograniczonym zakresie.

Przykładem może być aplikacja logvrp dostępna na stronie logvrp.com. Ekran główny aplikacji pokazany jest na Rys. 4.



Rys. 4 Zrzut ekranu systemu logvrp.com

Jej darmowa wersja jest ograniczona do 10 odbiorców. Możliwe jest rozwiązywanie szeregu różnych wariacji marszrutyzacji. W aplikacji zostały zaimplementowane dwa algorytmy do wyboru – zmodyfikowaną wersję algorytmu Jana Dethloffa oraz adaptacyjne wyszukiwanie sąsiadów w dużej skali.

## Instancje testowe

Istnieje wiele ogólnodostępnych instancji testowych do sprawdzenia poprawności rozwiązania proponowanego przez zaimplementowany algorytm. Dostępne są one na stronie internetowej [7].

### Plik \*.vrp

Dostępne one są do pobrania w postaci pliku \*.*vrp*. Plik ten ma jasno określoną strukturę. Można podzielić go na dwie części. Pierwsza z nich jest to specyfikacja problemu. Kolejna cześć natomiast mieści informację na temat konkretnych danych – położenia oraz wymagać odbiorców oraz magazynu.

### Specyfikacja problemu w pliku

Wszystkie wpisy w tej sekcji składają się z formy *<klucz>*: <*wartość>*, gdzie klucze mogą być następujące:

* *NAME* – identyfikator instancji
* *TYPE* – definiuje, jaki typ danych zawiera sekcja z danymi
* *COMMENT* – dodatkowy komentarz
* *DIMENSION* – jest to informacja na temat ilości odbiorców oraz magazynów
* *CAPACITY* – dla problemu klasy CVRP mówi o pojemności pojazdu
* *EDGE\_WEIGHT\_TYPE* – definiuje, w jaki sposób obliczana jest odległość pomiędzy odbiorcami i magazynem
* *EDGE\_WEIGHT\_FORMAT* – definiuje format odległości pomiędzy odbiorcami oraz magazynem, jeżeli takowe zostały podane, może być to funkcja lub gotowa macierz odległości
* *EDGE\_DATA\_FORMAT* – definiuje format połączeń pomiędzy wierzchołkami (magazynem oraz odbiorcami) zostały podane, jako lista lub macierz sąsiedztwa
* *NODE\_COORD\_TYPE* – definiuje czy i w jakim formacie koordynaty powiązane są z konkretnym wierzchołkiem – dane te mogą być wykorzystane do wyświetlenia graficznego rozwiązania bądź obliczenia odległości
* *DISPLAY\_DATA\_TYPE* – specyfikuje, w jaki sposób graficzna reprezentacja problemu może zostać uzyskana
* *EOF* – definiuje koniec pliku

### Część zawierająca dane

Tę część pliku można podzielić na podsekcje. W zależności od wcześniej podanych wartości w części ze specyfikacją mogą one mieć różne wartości. Najważniejszymi sekcjami są:

* *NODE\_COORD\_SECTION* – podane są koordynaty każdego z wierzchołków
* *DEPOT\_SECTION* – podana jest lista magazynów
* *DEMAND\_SECTION* – dla problemu klasy CVRP podane są wartości wymagań poszczególnych odbiorców
* *EDGE\_DATA\_SECTION* – zawiera informacje na temat odległości pomiędzy poszczególnymi wierzchołkami

# Projekt systemu

W tym rozdziale wyspecyfikowano wszystkie wymagania dotyczące systemu, te funkcjonalne oraz niefunkcjonalne. Następnie został przedstawiony zaprojektowany model danych wykorzystywany w aplikacji.

## Specyfikacja wymagań

Podczas projektowania systemu informatycznego, na samym początku warto wyznaczyć z góry, jakie wymagania powinien obejmować. Te można podzielić na dwie kategorie. Pierwsza z nich są wymagania funkcjonalne, opisują one funkcje, jakie ma realizować system z punktu widzenia użytkownika. Drugą z nich są wymagania niefunkcjonalne, w nich znajdują inne wymagania, często nazywane jakościowymi.

### Wymagania funkcjonalne

Tworzony system ma wpierać marszrutyzację pojazdów. Zakres wymagań funkcjonalnych, które powinna pokrywać aplikacja został przedstawiony na diagramie przypadków użycia przedstawionym na Rys 5.



Rys. 5 Diagram przypadków użycia

Aplikacja ta ma spełniać szereg wymagań:

* Logowanie do systemu – system ma wspierać logowanie się użytkownika do systemu. Dzięki czemu możliwe będzie zdalne zapisywanie pracy oraz wyników. Logowanie do systemu odbywało się będzie poprzez podanie loginu oraz hasła.
* Rejestracja użytkownika – użytkownik przed korzystaniem z aplikacji będzie miał możliwość założenia konta. Odbywało się to będzie przez podanie swojego loginu oraz hasła na odpowiedniej formatce w aplikacji.
* Zmiana hasła użytkownika – system powinien dawać możliwość edycji hasła przez użytkownika
* Edycja problemu – przez problem rozumie się wszystkich odbiorców oraz magazyn. System powinien dawać możliwość edycji poprzez interaktywną mapę. W ramach edycji problemu zawiera się:
  + Dodawanie magazynu lub odbiorcy – system powinien oferować dodawanie magazynu poprzez zaznaczenie miejsca na mapie oraz padaniu nazwy
  + Edycja magazynu lub odbiorcy – po wybraniu konkretnego magazynu lub odbiorcy system powinien dawać możliwość edycji jego parametrów

- w przypadku odbiorcy – jego położenie, nazwa oraz wymagana ilość towaru

- w przypadku magazynu – jego położenie oraz nazwę

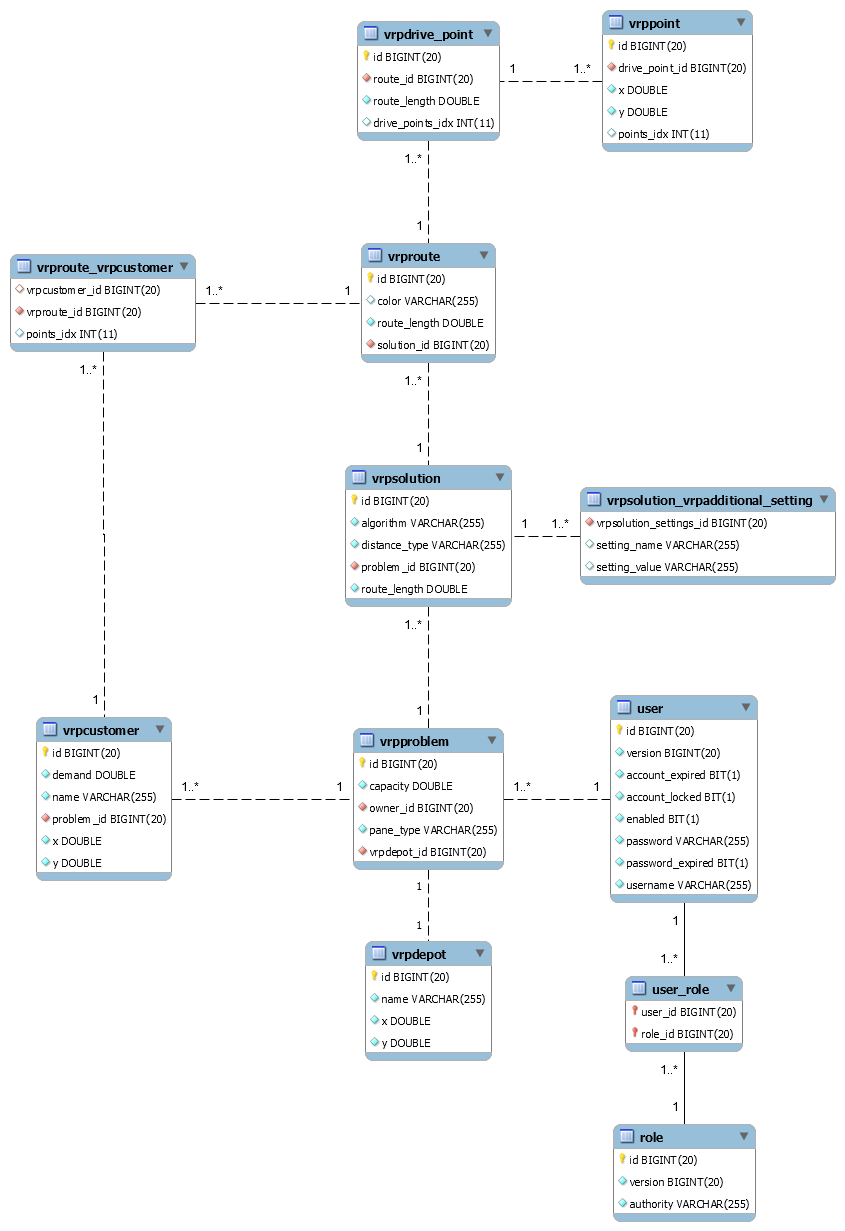
* Obliczanie rozwiązania – z wcześniej ustalonych danych aplikacja powinna dawać możliwość obliczenia rozwiązania po ustaleniu niezbędnych parametrów, takich jak sposób rozwiązywania.
* Wyświetlanie szczegółów rozwiązania – wcześniej obliczone rozwiązania powinno zostać przedstawione na mapie wraz z zaznaczonymi pozycjami magazynu oraz odbiorców. Jako, że rozwiązanie może składać się z wielu tras – każda z tras powinna mieć możliwość podświetlenia jej oraz wyświetlenia szczegółowych informacji na jej temat.
* Do zadanego problemu będzie można wyznaczyć kilka rozwiązań oraz porównać je między sobą. Między innymi poprzez możliwość sortowania pod względem długości.
* Zapisywanie problemów wraz z rozwiązaniami – użytkownik będzie mógł zapisać swoje rozwiązanie w bazie danych, lub też wyeksportować go do pliku.
* Import plików \*.vrp – w celu weryfikacji, jakości proponowanych sposobów rozwiązywania system powinien dać możliwość importowania plików w formacie vrp*.* Budowa plików została przedstawiona w poprzednim rozdziale.

### Wymagania niefunkcjonalne

System powinien spełniać szereg wymagać niefunkcjonalnych. To jest takich, które obejmują cechy tworzonego systemu, ale nie mają bezpośredniego wpływu nad oferowane możliwości dla użytkownika. Mimo to są bardzo ważne przy działaniu aplikacji.

* System ten powinien być możliwy do uruchomienia w przeglądarce internetowej, powinien być przez to dostępny na wszystkich systemach operacyjnych
* System powinien być rozszerzalny. Między innymi dzięki łatwej możliwości implementacji dodatkowych algorytmów.
* System powinien pokazywać informacje na temat obliczeń, jeżeli te trwają długo.
* System musi wykorzystywać tylko darmowe biblioteki oraz platformy programistyczne

## Struktura bazy danych

Szczegółowy diagram reprezentujący schemat bazy danych jest pokazany na Rys. 6.

Rys. 6 Diagram ERD pokazujący model danych

Na diagramie możemy zauważyć tabele odpowiedzialne za przechowywanie informacji na temat użytkowników. Mowa o tabelach ‘user’, ‘user\_role’, ‘role’. W nich trzymane są informacje na temat konta użytkownika jak i jego hasło w zaszyfrowanym formacie. Każdy użytkownik może być przypisany do dowolnej ilości ról.

Użytkownik jest w relacji jeden do wielu z tabelą ‘vrpproblem’. Oznacza to, że użytkownik może posiadać wiele problemów. Do problemu też jest przypisany maksymalna pojemność pojazdów, magazyn, którego dane trzymane są w osobnej tabeli (‘vrpdepot’) oraz wiele magazynów. Każdy magazyn oprócz koordynat posiada nazwę oraz wymaganą ilość towaru.

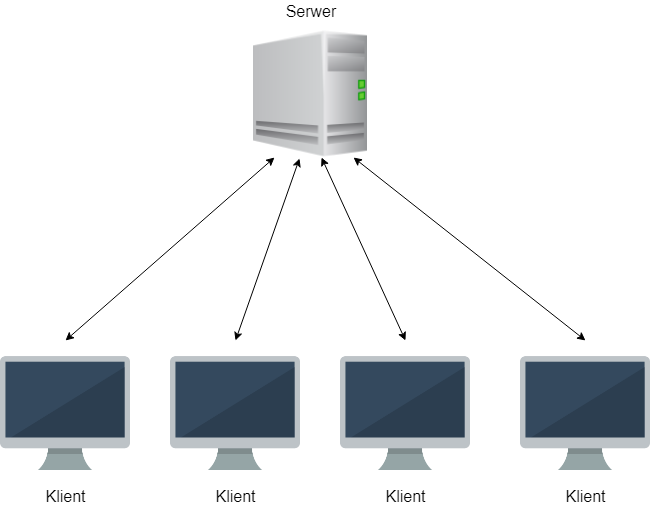
Każdy problem może mieć wiele rozwiązań, są one przedstawione w tabeli ‘vrpsolution’. W której zapisane jest między innymi wybrany algorytm wraz z swoimi ustawieniami. Jako że każdy z algorytmów może mieć inny zestaw ustawień, zdecydowano na utworzenie specjalnej tabeli (‘vrpsolution\_vrpadditional\_setting’), która oprócz klucza głównego posiada także nazwę i wartość ustawienia.

Do rozwiązania przypisane jest wiele tras, każda z tras ma kolejno ustawionych odbiorców, przestawia to tabela ‘vrproute\_vrpcustomer’. Pole ‘points\_idx’ w tej tabeli przetrzymuje numer odbiorcy, który definiuje, w jakiej kolejności zostaną odwiedzone poszczególne punkty.

W systemie też przetrzymywane są informacje na temat szczegółowych tras pojazdu. Dane te przetrzymywane są w formie krótszych odcinków, które natomiast są definiowane poprzez konkretne punkty na mapie.

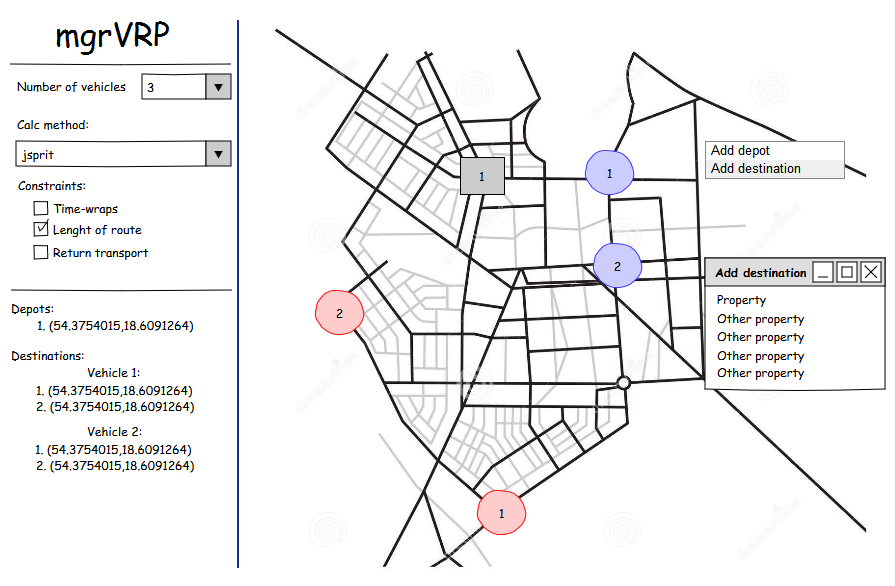
## Planowana architektura systemu

Z racji tego, że budowany system internetowy ma być dostępny przez przeglądarkę oraz wpierać manipulację mapą, z drugiej strony natomiast musi przeprowadzać zaawansowane obliczenia, które niekiedy mogą być czasochłonne zdecydowano, podzielić system na dwie główne składowe części. Ogólna architektura systemu została przedstawiona na diagramie poniżej (Rys. 7).



Rys. 7 Planowana architektura systemu

Po stronie klienta zostanie pobrana aplikacja, która uwierzytelni się w centralnym serwerze. Po pomyślnym uwierzytelnieniu zostanie uruchomiona aplikacja. Początkowy zarys planowanego wyglądu przedstawia Rys. 8.

******

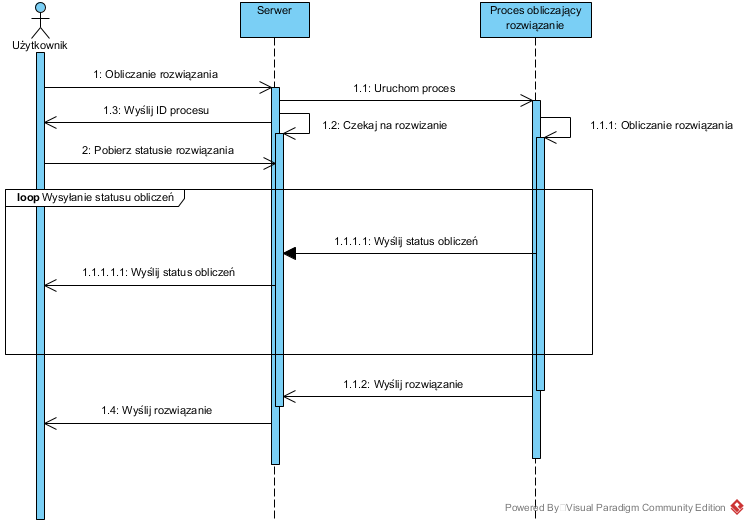
Rys. 8 Planowany zarys aplikacji klienckiej

Planowano, aby centralna część aplikacji oraz główny jej obszar zajmowała mapa. Natomiast po lewej stronie można byłoby ustawiać poszczególne parametry. Końcowy aplikacja znacznie odbiega od planów, jednak rozmieszczenie elementów pozostało tak jak projektowano.

Z drugiej strony planowane jest też utworzenie aplikacji po stronie serwera, będzie ona odpowiedzialna za główne obliczenia oraz zarządzanie użytkownikami oraz ich dostępem do systemu.

## Diagram procesu obliczania rozwiązania

Proces obliczania rozwiązania przez serwer jest ukazany jest na Rys. 9.



Rys. 9 Diagram sekwencji przedstawiający proces obliczania rozwiązania

Po otrzymaniu żądania serwer na początku sprawdza uprawnienia zalogowanego użytkownika oraz waliduje rozwiązanie. Jeżeli problem pomyślnie ja przejdzie. Problem ten jest przesyłany dalej do serwisu odpowiedzianego za generowanie unikalnego numeru ID.

Następnie system tworzy współdzieloną kolejkę zdarzeń oraz przypisuje jej wcześniej wygenerowany numer.

W następnej kolejności tworzony jest proces w ramach, którego wykonują się obliczenia. Podczas obliczeń do użytkownika mogą zostać wysłane następujące wiadomości:

* Zwykła tekstowa wiadomość informująca użytkownika o statusie obliczeń, na przykład o tym, że macierz odległości została już obliczona, i proces zaczął właściwe obliczanie wyniku.
* Częściowe rozwiązanie, niektóre algorytmy – na przykład algorytm Tabu Search, który po każdej iteracji generuje całkowite rozwiązanie, które później jest ulepszane. Może ono zostać wyświetlone użytkownikowi.
* Końcowe rozwiązanie – każdy proces kończy się zwracając rozwiązanie.
* Błąd – komunikat informujący o błędzie podczas obliczeń. Podobnie do końcowego rozwiązania po jego wystąpieniu algorytm kończy swoje działanie a do użytkownika nie jest wysyłane rozwiązanie.

Gdy proces generuje wiadomość dla użytkownika jest ona wpisywana do kolejki zdarzeń.

W zależności od wybranego sposobu komunikacji z serwerem albo to system inicjuje wysłanie informacji o tym, że do kolejki zdarzeń został dodany nowy wpis. W takim przypadku pobiera oraz usuwa z kolejki najstarszy wpis oraz wysyła do użytkownika. W innym wypadku to użytkownik może odpytywać serwer o to czy w kolejce pojawiło się nowe rozwiązanie, jeżeli tak to serwer pobiera i usuwa z kolejki najstarsze zdarzenie oraz wysyła je użytkownika.

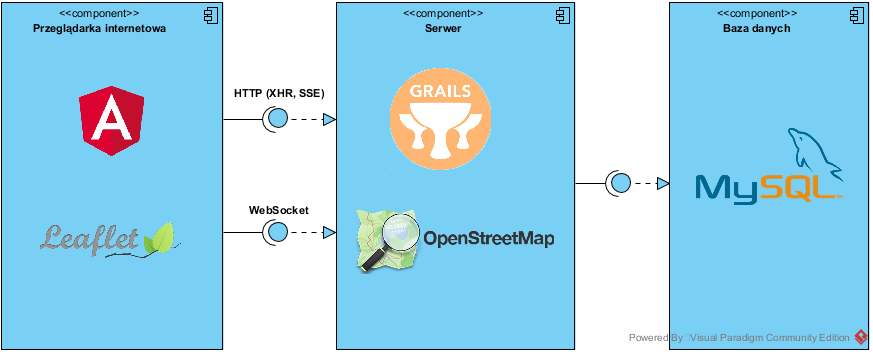
Dzięki oddelegowaniu obliczeń do osobnego procesu uzyskane została możliwość wysyłania statusu obliczeń do użytkownika oraz oddzielenie części komunikacyjnej oraz obliczeń. Dodatkowo w przypadku utraty połączenia pomiędzy użytkownikiem oraz serwerem użytkownik może pobrać informacje na temat rozwiązania w późniejszym czasie.

# Opis systemu

## Architektura

Stworzony system został oparty na architekturze klient-serwer. W ramach pracy powstały dwie aplikacje. Pierwsza z nich jest to aplikacja kliencka. Pozwala ona na graficzną obsługę mapy, edycję parametrów obliczeń wyświetlania wyników. Z drugiej strony natomiast stworzono aplikację webową możliwą na oddzielną instalację. Odpowiedzialna jest ona za wszystkie obliczenia, autoryzację użytkownika oraz zapisywanie danych do bazy.

Na diagramie poniżej (Rys. 10) przestawione zostały wykorzystane technologie w poszczególnych aplikacjach oraz schemat komunikacji między nimi. Poszczególne części będą opisane w kolejnych podrozdziałach.



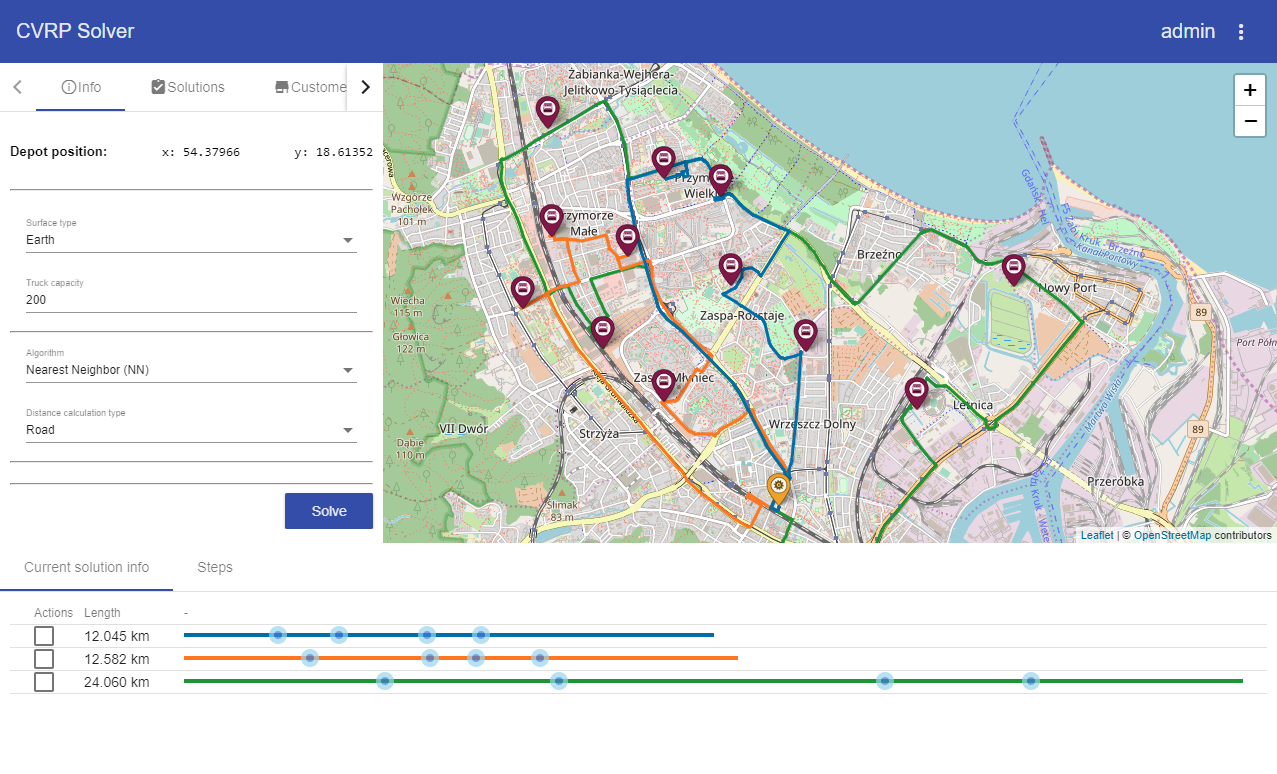
Rys. 10 Diagram przedstawiający architekturę aplikacji

## Część kliencka

Aplikacja uruchamiana w przeglądarce internetowej jest odpowiedzialna za pobieranie danych od użytkownika oraz przekazywaniu ich do części serwerowej w celu opracowania wyników. Z tego powodu powinien być on jak najprostszy oraz w intuicyjny sposób dawać możliwość na edycję danych. Część ta została zbudowana w podejściu SPA (Single Page Application) [8]. Oznacza to, że strona, która została załadowana na początku, po interakcji użytkownika nie odświeża całej strony z serwera tylko dynamicznie zmienia aktualną stronę. Dzięki temu użytkownik ma wrażenie jakby korzystał z natywnej aplikacji.

W tym celu do stworzenia aplikacji została wykorzystana platforma programistyczna Angular [9] w wersji 4.2.4. Jest to platforma utworzona oraz wpierana i rozwijana przez firmę Google. Utworzona aplikacja została napisana przy użyciu języka TypeScript.

Ekran startowy aplikacji można zobaczyć na Rys. 11.



Rys. 11 Ekran główny aplikacji

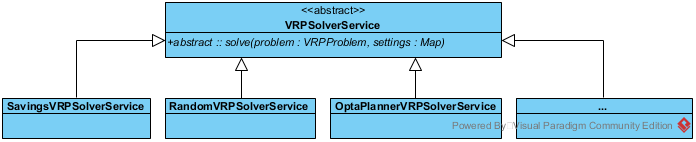
Program napisany przy użyciu Angulara dzieli się na moduły, te natomiast na komponenty. Jednym z takich utworzonych komponentów jest komponent mapy. Zajmuje się on wyświetlaniem wszystkich informacji na mapie. Do obsługi mapy wykorzystano bibliotekę leaflet.js [10] w wersji 1.3.1. Jest ona napisana przy użyciu JavaScript. Pozwala ona na wyświetlanie mapy w postaci kafelek. W tworzonej aplikacji kafelki zostały pobierane z serwisu: tile.osm.org. Zawiera ona mapę generowaną na podstawie danych z projektu OpenStreetMap.

## Część serwerowa

Serwer w głównej mierze odpowiedzialna jest za obliczanie rozwiązania dla zadanego problemu. Schemat działania został przedstawiony w poprzednim rozdziale.

Część ta została utworzona przy użyciu platformy programistycznej Grails w wersji 3.2. Implementuje ona wzorzec architektoniczny Model-Widok-Kontroler. Dzięki czemu możemy podzielić ją na trzy części: model, widok i kontroler.

Kontroler przyjmuje zapytania od użytkownika. Aplikacja korzysta głównie z jednego kontrolera o nazwie VRPController, w którym zaimplementowane są wszystkie niezbędne metody. Po otrzymaniu żądania rozwiązania problemu kontrola przekazywana jest do odpowiedniego serwisu, który uruchamia proces rozwiązywania używając jednego z dostępnych metod rozwiazywania. Każda z zaimplementowanych metod rozwiązywania jest reprezentowana przez osobną klasę rozszerzającą abstrakcyjną klasę *VRPSolverService.* Przykładowe zaimplementowane sposoby pokazane są na Rys. 12.



Rys. 12 Diagram klas

Grails został stworzony w podejściu konwencja ponad konfiguracją. Dzięki czemu nie musimy podawać, że stworzyliśmy nową klasę serwisową odpowiedzialną za rozwiązywanie problemu. Zostanie on automatycznie dodany, jako globalny serwis, który może być używany w aplikacji. Podczas przekierowywania obsługi problemu do konkretnej implementacji są przeszukiwane wszystkie zaimplementowane serwisy, które rozszerzają klasę bazową *VRPSolverService* i wybierany ten z odpowiednia nazwą.

Jak opisano w rozdziale pierwszym, istnieją różne sposoby na definiowanie odległości pomiędzy poszczególnymi punktami na mapie. W aplikacji zostały wybrane dwie z nich, odległość drogowa (obliczanie najkrótszej trasy samochodowej) oraz lotnicza (bezpośrednia).

W celu obliczenia odległości drogowej potrzebne są dane na temat dróg. Tutaj podobnie jak w wypadku części klienckiej wykorzystano mapy OpenStreetMap [8]. Dane na temat dróg można pobrać ze strony download.geofabrik.de. Po pobraniu map należy ich lokalizację w pliku konfiguracyjnym. Ten znajduje się w katalogu */grails-app/conf* o nazwie *application.yml*. Należy wskazać lokalizację map oraz katalogu tymczasowego kolejno podając wartości wpisów *mappath* oraz *mappathtmp*. Za obliczanie odległości drogowej pomiędzy dwoma punktami odpowiedzialna jest klasa *GraphHopperOSMService.* Korzysta ona z biblioteki GraphHopper.

Zaimplementowano dwa algorytmy obliczające rozwiązanie. Jednym z nich jest algorytm oszczędności natomiast kolejnym jest algorytm najbliższego sąsiada. Utworzono też serwis odpowiedzialny za generowanie losowego rozwiązania do problemu.

Do aplikacji zostały też dodane dwie biblioteki. Pierwszą z nich jest Jsprit. Biblioteka ta wykorzystuje inny format danych wejściowych do problemu, dlatego też wymagana jest translacja z jednego formatu na drugi. Wszystkie te operacje oraz wywołanie biblioteki odbywają się w klasie *JspritVRPSolverService*.

Drugą biblioteką było Google Optimization Tools (Google OT). Tutaj sytuacja była bardziej skomplikowana. Niestety cała biblioteka udostępniona jest w formacie dll. Wcześniej musimy dołączyć ją, jako bibliotekę systemową. Umożliwia nam to metoda *loadLibrary* z klasy *System*. Wywołujemy ją następująco: *System.loadLibrary("jniortools")*, wcześniej jednak podczas uruchamiania programu musimy ustawić zmienną konfiguracyjną *java.library.path* na katalog, w którym istnieje plik dll.

## Budowanie oraz instalacja aplikacji

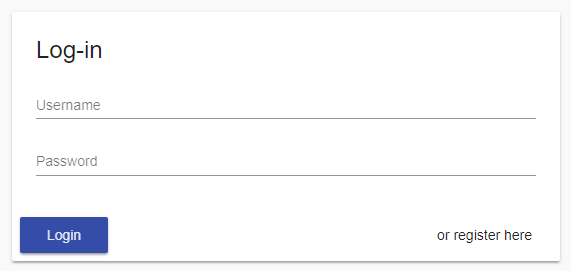
Obydwie części – część serwerowa oraz kliencka muszą zostać zbudowane oraz zainstalowane osobno.

Budowanie części klienckiej odbywa się przy użyciu narzędzia Angular CLI. W celu zbudowania aplikacji należy uruchomić polecenie *ng build.* W pierwszej kolejności pobiera on wszystkie niezbędne zależności (zdefiniowane w pliku *package.json*), następnie transkompiluje kod napisany w języku TypeScript na język JavaScript. Dodatkowo kopiuje wszystkie niezbędne pliki(obrazy, style, skompilowany kod) do katalogu *dist*. Tak zbudowana aplikacja może zostać zainstalowana na serwerze HTTP – na przykład Apache HTTP Server.

Część serwerowa natomiast budowana jest przy użyciu systemu Gradle. Aby zbudować aplikację należy w katalogu głównym uruchomić skrypt: *gradlew war*. Spowodowuje on podobnie pobranie wszystkich zależności zdefiniowanych w pliku *build.gradle* w sekcji *dependencies*. Później skompiluje kod aplikacji. Gotowa aplikacja dostępna będzie w katalogu */build/libs* w pliku *api\*.war*. Plik ten może zostać zainstalowany na dowolnym kontenerze aplikacji webowych – na przykład Apache Tomcat.

## Opis interfejsu użytkownika

Po otworzeniu aplikacji ukarze się ekran logowania użytkownika (Rys. 13).



Rys. 13 Ekran logowania aplikacji

Jeżeli nie mamy jeszcze konta w systemie należy je założyć. Po poprawnym zalogowaniu się ukaże się główny ekran aplikacji. Można go podzielić na 4 główne komponenty:

* Górny pasek wyświetlający nazwę aplikacji oraz nazwę zalogowanego użytkownika.
* Komponent, który pozwala na zarządzanie oraz wyświetlanie ustawień oraz statusu obliczeń.
* Mapę, która wyświetla rozmieszczenie oraz obliczone rozwiązanie.
* Dolne komponent wyświetlający szczegółowe informacje na temat aktualnie wybranego rozwiązania.

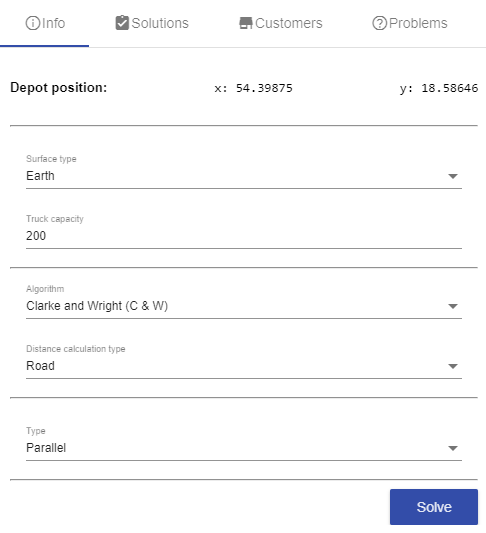
### Górny pasek aplikacji

Oprócz wyświetlania nazwy użytkownika, możemy po prawej stronie otworzyć menu, które pozwoli nam na wykonanie kilku akcji. Pierwszą z nich jest zapisanie danych z systemu. Po wybraniu tej opcji wszystkie dane zostaną zapisane w bazie danych, dzięki temu będziemy mieli do nich dostęp w późniejszym czasie. Kolejnym elementem w menu są ustawienia, pozwalają one na wybranie sposobu komunikacji z serwerem oraz adres serwera. Dalsze elementy menu odpowiedzialne są za zarządzanie zalogowanym użytkownikiem – wylogowanie oraz zmiana hasła.

### Komponent zarządzający

Komponent ten znajduje się po lewej stronie od mapy. Pozwala on na centralne zarządzanie systemem. Podzielony został na cztery zakładki.

Pierwszą z nich jest „Info”. Jej wygląd można zobaczyć na Rys. 14.



Rys. 14 Zakładka „Info”

Zawiera ona informacje na temat położenia magazynu oraz menu konfiguracyjne pozwalające na sparametryzowanie obliczeń.

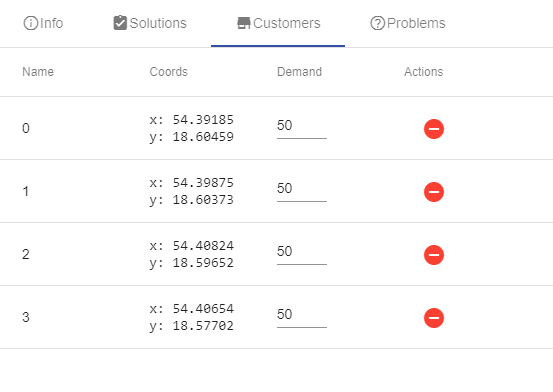
Pierwszym z nich jest pole wyboru „Sufrace type” pozwalające na wybranie jednej z dwóch opcji:

* Earth – typ mapy zostanie zmieniony na mapę ziemi
* Simple – typ mapy zostanie zmieniony na siatkę

Wybór ten będzie miał też wpływ na sposób obliczania odległości pomiędzy punkami na mapie przez system.

Kolejną elementem jest możliwość ustawienia maksymalnej pojemności pojazdu. Dalej, w kolejnej sekcji możemy wybrać jeden z zaimplementowanych algorytmów lub metod rozwiązywania zadanego problemu. Pole „Distance calculation type” dostępne jest dla typu mapy „Earth”, definiuje ono sposób obliczania odległości pomiędzy punktami – dostępne są dwie możliwości: „Air”, po której wybraniu obliczane będą odległości w linii prostej oraz „Road” po wybraniu, której odległość będzie obliczana na podstawie danych drogowych.

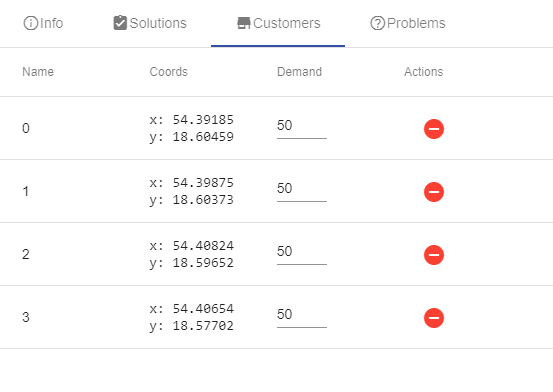
Po poprawnym uzupełnieniu danych oraz wybraniu odpowiednich proces obliczania można rozpocząć poprzez naciśnięcie przycisku „Solve”. Jako, że proces ten może trwać długo, wpis o rozpoczętym procesie możemy zobaczyć na zakładce ‘Solutions’ pokazanej na Rys. 15.



Rys. 15 Zakładka „Solutions”

Tę zakładkę możemy podzielić na dwie części pierwszą z nich jest wyświetlanie wcześniej obliczonych rozwiązań w postaci tabeli. Rozwiązania te możemy posortować pod względem długości tras. Na końcu każdego wiersza znajduje się komórka z dostępnymi akcjami. Możemy wyświetlić szczegóły konkretnego rozwiązania, zostanie ono też wyświetlone na mapie, lub możemy usunąć rozwiązanie z systemu. Kolejną częścią tej zakładki jest lista aktualnie obliczanych rozwiązań, wyświetlane tam zostaje aktualny status obliczeń oraz wybrane ustawienia obliczania.

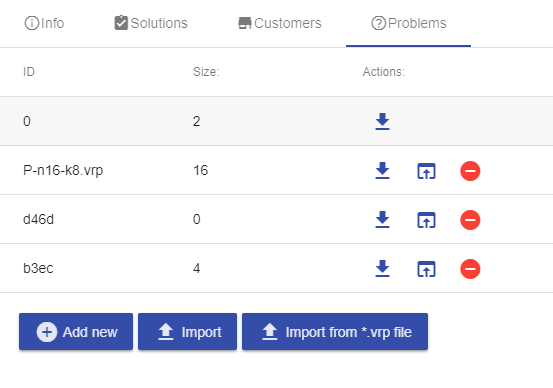
Trzecią zakładką jest zakładka wyświetlająca wszystkich odbiorców. Została ona pokazana na Rys. 16.



Rys. 16 Zakładka „Customers”

Posiada ona tabelę wyświetlającą wszystkie informacje na temat odbiorców. Pozwala na usunięcie wybranego odbiorcy bądź też wyświetlenie szczegółowych danych na jego temat.

Ostatnią zakładką jest „Problems” pokazana na Rys. 11.



Rys. 17 Zakładka „Problems”

Aplikacja w tym samym momencie może pracować na kilku problemach. Można się dowolnie pomiędzy nimi przełączać. Na powyższym rysunku można zobaczyć 3 problemy wyświetlone w tabeli. Pierwsza kolumna w tej tabeli oznacza unikalne ID problemu. Następna zatytułowana „Size” oznacza liczbę odbiorców w danym problemie. Ostatnią kolumną są akcje, które są dostępne na podanym problemie.

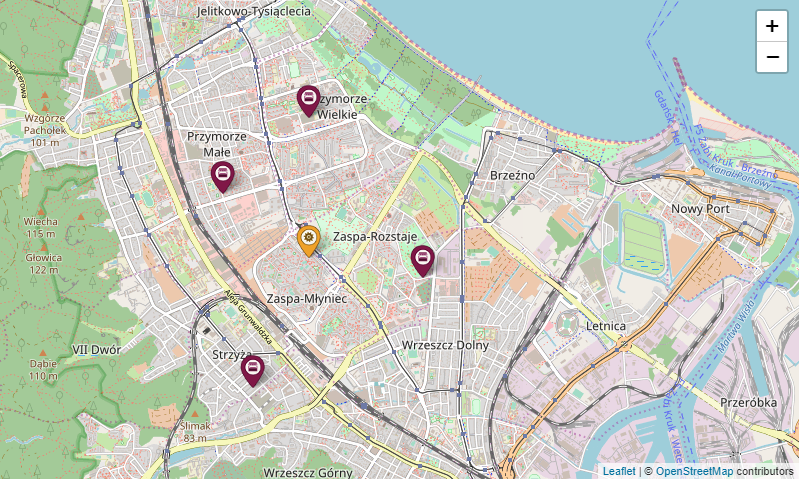
Pierwszą z nich jest export. Problem eksportowany jest w formacie *json*. W wygenerowanym pliku znajdują się wszystkie ustawienia oraz wcześniej obliczone rozwiązania.

Kolejną akcją jest załadowanie problemu. Opcja ta nie jest dostępna dla pierwszego z nich, ponieważ jest ona aktualnie wczytany.

Pod tabelą znajdują się trzy przyciski. Przycisk o nazwie „Add new” odpowiedziany jest za dodawanie nowego problemu z domyślnymi ustawieniami. Kolejny przycisk - „Import”. Pozwala na wczytanie wcześniej wyeksportowanych problemów w formacie \*.json. Ostatnim przyciskiem dostępnym na tej zakładce jest „Load \*.vrp file”, który pozwala na wczytanie problemu z pliku \*.vrp.

### Mapa

Centralnym komponentem w aplikacji jest mapa wyświetlona po prawej stronie ekranu. Mapa ta może być wyświetlana w dwóch trybach. Domyślnie wyświetlana jest mapa ziemi. Przedstawiono to na Rys. 18. Problem może być przedstawiony niekoniecznie, jako współrzędne na mapie, lecz punkt na przestrzeni dwuwymiarowej. Dlatego program daje możliwość przełączenia tła, w którym zamiast mapy wyświetlana jest siatkę z wymiarami (Rys. 19).



Rys. 18 Mapa z zaznaczonymi odbiorcami oraz magazynem

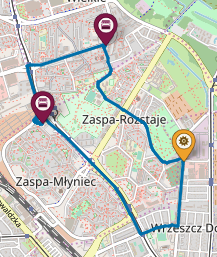


Rys. 19 Siatka wraz z zaznaczonymi punktami

Klikając na mapę, w danym miejscu możemy dodać magazyn lub odbiorcę. Odbiorca na mapie zaznaczony jest symbolem pokazanym na Rys. 20 natomiast magazyn identyfikuje ikona pokazana na Rys. 21.

|  |  |
| --- | --- |
| Rys. 20 Ikona odbiorcy | Rys. 21 Ikona magazynu |

Ikony te można przenosić metodą przeciągnij i upuść, w ten sposób możemy zmienić ich położenie. Natomiast klikając w ikonę odbiorcy możemy wyświetlić szczegółowe informacje na jego temat. Na mapie mogą być również wyświetlane trasy, każda z tras zaznaczona jest osobnym kolorem. Przykładowa trasa pokazana jest na Rys. 22.

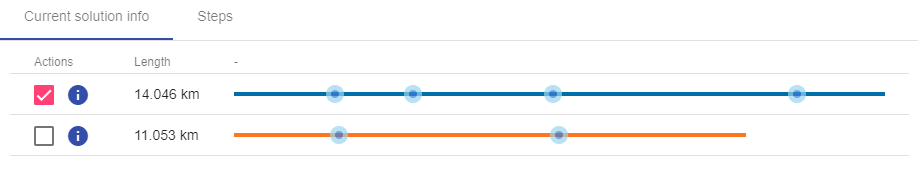


Rys. 22 Trasa łącząca dwóch odbiorców oraz magazyn

Po kliknięciu na trasę wyświetlone zostają szczegóły na jej temat.

### Komponent wyświetlający szczegóły rozwiązania

Komponent ten wyświetlany jest na dole aplikacji. Posiada on dwie zakładki, pierwsza z nich zatytułowana „Current solution info” wyświetla listę wszystkich tras. Pierwsza kolumna odpowiedzialna jest za akcje, można wyświetlić szczegóły na temat trasy lub też ją ukryć na mapie. Dalej wyświetlana jest jej długość oraz wyświetlana w sposób graficzny jej odwzorowanie wraz z zaznaczonymi odbiorcami. Przykładowy widok można zobaczyć na Rys. 23.



Rys. 23 Wyświetlone szczegóły trasy

Na zakładce „Steps” dostępna jest liska kroków, które zostały wykonane do obliczenia danego rozwiązania.

## Komunikacja

Proces obliczania rozwiązania może być czasochłonny. Istnieje kilka sposobów komunikacji z serwerem w celu ustalenia czy rozwiązanie zostało już obliczone lub na jakim etapie właśnie się proces obliczający znajduje. Sposoby te zostaną poniżej opisane oraz porównane pod względem szybkości działania oraz ilości przesyłania danych. Zostaną także przedstawione zalety oraz wady każdego z użytych rozwiązań.

### XHR

Protokół XMLHttpRequest (XHR) [11] jest interfejsem API pozwalającym przesyłać informację do i z serwera za pomocą JavaScript. Dzięki XHR nie jest konieczne odświeżenie strony WWW w celu jej aktualizacji jednak można to zrobić asynchronicznie pod kontrolą kodu JavaScritpt. Protokół ten nie tylko udostępnia komunikację, ale również ją znacząco upraszcza. Przeglądarka za programistę zajmuje się wieloma aspektami: między innymi zarządzaniem połączenia, negocjacją protokołów lub formatowaniem zapytań HTTP. Za pomocą XHR możemy przesyłać dane zarówno binarne jak i tekstowe.

var xhr = new XMLHttpRequest();

xhr.open('GET', 'url');

xhr.responseType = 'arraybuffer';

xhr.onload = function() {

if (this.status == 200) {

var arraybuffer = this.response;

}

};

xhr.send();

Jak pokazuje powyższy przykład, na początku tworzymy obiekt XMLHttpRequest a następnie ustawiamy typ zwracanych danych, jako „arraybuffer”.

Protokół XHR potrafi skutecznie pobierać aktualizacje z serwera. Klient wysyła żądanie, a następnie serwer na nie odpowiada. Nie istnieje jednak możliwość inicjowania połączenia w odwrotnym kierunku.

### Odpytywanie w protokole XHR

Jedną ze strategii odbierania aktualizacji z serwera jest okresowe wysyłanie zapytania do serwera w regularnych odstępach czasu. Jeżeli na serwerze pojawi się odpowiedz to jest ona wysyłana, w przeciwnym wypadku odsyłana jest pusta wiadomość. Jest to bardzo proste w implementacji. Ważną kwestią jest ustawienie odpowiedniego interwału, zbyt krótki generowałby zbędny ruch orz obciążenie klienta oraz serwera, natomiast zbyt długi opóźnione aktualizacje.

function getUpdates(url){

var xhr = new XMLHttpRequest();

xhr.open(‘GET’, url);

xhr.onload = function(){ … };

xhr.send();

}

setInterval(getUpdates(‘/vrp/solver/12345’), 500);

Na powyższym przykładzie widać okresowe wysyłanie zapytania do serwera, co pół sekundy.

Kolejną możliwością jest długotrwałe odpytywanie w protokole XHR. Modyfikując tak ostatni sposób, aby nie zwracać pustej odpowiedzi a zamiast tego utrzymywać otwarte połączenie aż do pojawienia się aktualizacji. Dzięki temu klient może otrzymać odpowiedź tak szybko jak pojawi się aktualizacja. Dzięki zastosowaniu takiej techniki zmniejsza się narzut wprowadzany przez odpytywanie.

function getUpdates(url){

var xhr = new XMLHttpRequest();

xhr.open(‘GET’, url);

xhr.onload = function () {

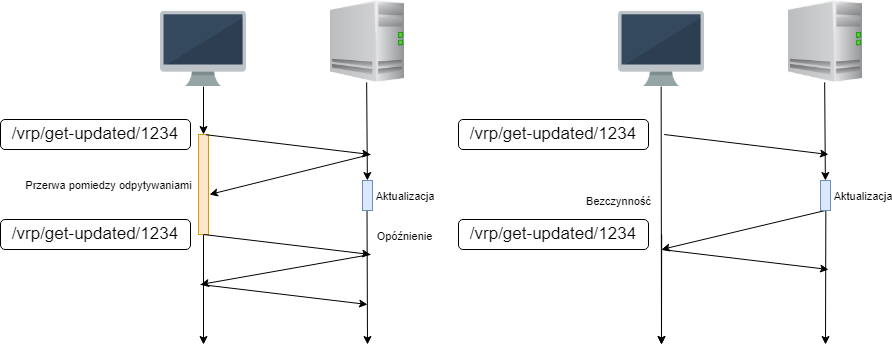
getUpdates(url);

}

}

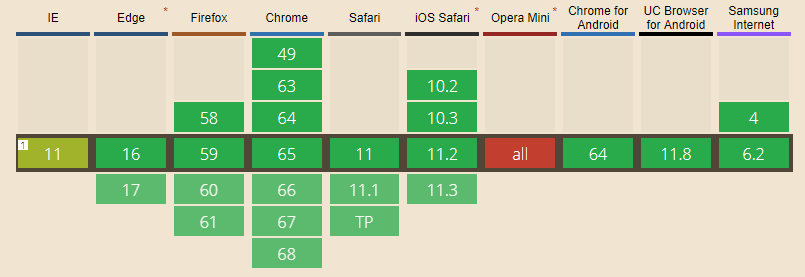
getUpdates(‘/vrp/solver/1234’);

Jak widać powyżej, po przetworzeniu odpowiedzi od serwera jest wysyłane nowe żądanie. Takie podejście posiada też wady, może generować większy ruch niż okresowe, jeżeli częstotliwość odpowiedzi z serwera jest bardzo duża. Dodatkowo utrzymanie każdego połączenia na serwerze jest kosztowne oraz wykorzystuje zasoby.



Rys. 24 Porównanie metod

Na Rys. 24 po lewej stronie możemy zobaczyć odpytywanie zwykłe oraz długotrwałe z prawej strony. Na Rys. 25 pokazany jest widok ze strony: caniuse.com/#feat=xhr2, na którym możemy zobaczyć, że prawie wszystkie przeglądarki obsługują w pełni protokół XHR2(oznaczone na zielono). Jedynym wyjątkiem jest Opera Mini oraz Internet Explorer w wersji 11, który nie obsługuje typu zwracanych danych, jako „json”.



Rys. 25 Przeglądarki obsługujące XHR

### SSE

Server-Sent Events jest standardem, który umożliwia pozwala na wysyłanie powiadomień z serwera do klienta w czasie rzeczywistym. Składa się z dwóch komponentów: typu danych „text/event-stream” oraz interfejs EventSource w przeglądarce.

var source = new EventSource("/vrp/solver/1234");

source.onopen = function () { … };

source.onerror = function () { … };

source.addEventListener("foo", function (event) { … });

source.onmessage = function (event) {

console.log(event.id, event.data);

if (event.id == "CLOSE") {

source.close();

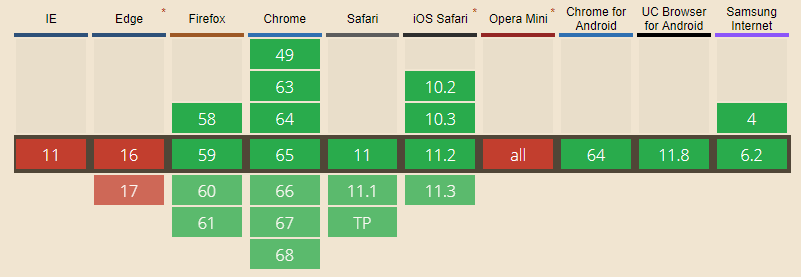
}

}

Na powyższym przykładzie widać na początku otworzenie połączenia do punktu końcowego połączenia. W kolejnych linijkach mamy obsługę zdarzeń ‘onload’ oraz ‘onerror’. Kolejna linijka przedstawia subskrypcję zdarzenia typu ‘foo’. Ostatnia metota jest wywoływana zawsze, niezależnie od zdarzenia, jakie wystąpiło. Identyfikator, typ oraz granice każdego komunikatu są definiowane przez protokół strumieniowania zdarzeń. Składa on się z predefiniowanych nazw pól:

* event: nazwa zdarzenia, takie jak ‘message’ lub ‘foo’
* data: pole zawierające dane
* retry: liczba definiująca czas ponownego połączenia w przypadku utracenia połączenia
* id: identyfikator wiadomości

Protokół SSE oferuje wbudowaną obsługę utraty przerwanych połączeń oraz ponowne wysłanie utraconych komunikatów. Jak widać na poniższym zżucie ekranu (Rys. 26) interfejs *EventSource* wspierany jest przez większość nowoczesnych przeglądarek. Wyjątkiem tutaj są przeglądarki Edge oraz Internet Explorer.



Rys. 26 Przeglądarki obsługujące Server-Sent Events

Dodatkowo protokół ten nie jest przystosowany do przesyłania danych binarnych. Ograniczenie to jednak można ominąć przy użyciu metody base64 do zakodowania obiektu binarnego. Dzięki temu jednak jest wprowadzany znaczny narzut(około 33% [12]).

### WebSocket

WebSocket jest protokołem wpierającym dwukierunkową transmisję danych binarnych lub tekstowych. Składa się z dwóch elementów [13]. Pierwszą z nich jest ustalenie parametrów transmisji danych na poziomie komunikatów HTTP. Kolejną zaś jest wymiana danych z użyciem niewielkiej wielkości ramek.

Do nawiązania połączenia WebSocket początkowo klient wysyła żądanie do serwera. Jest to zwykłe żądanie HTTP z dodatkowo ustawionymi nagłówkami. Przykładowe żądanie pokazane jest poniżej.

GET /chat HTTP/1.1

Host: server.pl

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: x3JJ3456345639GBhXDw==

Sec-WebSocket-Protocol: customprotocol

Sec-WebSocket-Version: 13

Origin: [http://](http://example.com)server.pl

Natomiast odpowiedź serwera wygląda następująco:

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

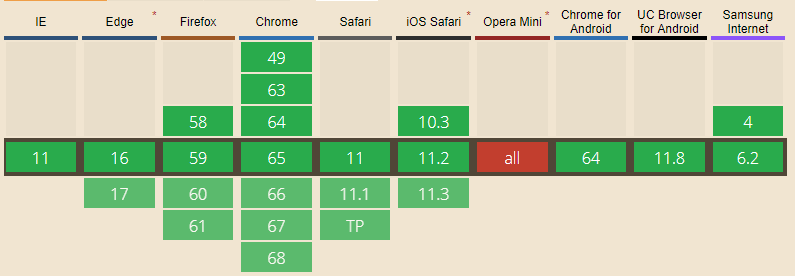
Connection: Upgrade

Sec-WebSocket-Accept: HSmrc0sM4567456745pG2HaGWk=

Sec-WebSocket-Protocol: customprotocol

Dzięki takiemu podejściu oba protokoły wykorzystują ten sam port TCP 80. Przykładowymi nagłówkami, które możemy użyć do negocjacji połączenia są:

* Sec-WebSocket-Version – definiuje wersję protokołu WebSocket
* Sec-WebSocket-Key – gnerowany przez klienta klucz, upewniający się, że serwer obsługuje daną wersję protokołu
* Sec-WebSocket-Accept – odpowiedź serwera na wartość klucza Sec-WebSocket-Key
* Sec-WebSocket-Protocol – negocjacja protokołu aplikacji
* Sec-WebSocket-Extensions – nagłówek wykorzystywany do negocjacji rozszerzeń protokołu WebSocket



Rys. 27 Przeglądarki obsługujące WebSocket

Protokół WebSocket jest wspierany przez wszystkie nowoczesne przeglądarki (z wyłączeniem Opery Mini). Analogicznie jak w poprzednich metodach jest to pokazane na zrzucie ekranu ze strony caniuse.com na Rys. 27.

### Porównanie

W ramach pracy zostały zaimplementowane wszystkie z wymienionych wyżej sposobów wymiany informacji z serwerem. Każde z nich posiada pewne wady. Jedynym pozytywnym aspektem wypowiadającym się za cyklicznym odpytywaniem XHR jest jego łatwość w implementacji.

W celu dodatkowego porównania wygenerowano test. Był to problem z losowo rozmieszczonymi odbiorcami oraz magazynem. Włączono rozwiązywanie tego problemu przez 5 algorytmów po kolei. Po każdym z nich do klienta została wysyłana informacja o aktualnym statusie obliczeń. Daje to 5 aktualizacji. Obliczenia po stronie serwera trwały około 30s. Dla odpytywania XHR interwał odpytywania ustawiono na 200ms. Przy pomocy programu Fiddler Web Debugger zmierzono ilość danych przesłanych pomiędzy serwerem a klientem. Wyniki pokazane są w Tabeli 3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Odpytywanie XHR** | **Kometa XHR** | **SSE** | **WebSocket** |
| **Narzut danych w bajtach** | 139188 | 3620 | 758 | 1078 |

Tabela 3 Porównanie narzutu danych dla różnych sposobów komunikacji

Metoda cyklicznego odpytywania XHR uzyskała najgorszy wynik z wszystkich metod komunikacji. W pozostałych przypadkach wartości były na podobnym poziomie. Mimo wszystko wszystkie wartości stanowiły bardzo mały procent przesyłanych wszystkich danych i nie stanowi to większego problemu. Dodatkowo warto zaznaczyć, że podczas obliczania metoda krótkiego odpytywania XHR wysłała 202 zapytania do serwera. Spowodowało to znaczne zużycie łącza oraz innych zasobów komputera.

Biorąc pod uwagę wszystkie cechy przedstawionych protokołów, najlepszym wyborem będzie wybranie komunikacji poprzez protokół SSE. Niestety nie jest on dostępny w niektórych starszych przeglądarkach. W takim przypadku warto w zamian użyć metody komety XHR, która dostępna jest dla wszystkich przeglądarek i stanowi dobrą alternatywę. Niestety w ten sposób pozbywamy się między innymi automatycznego ponawiania połączenia po rozłączeniu.

# Wyniki

Długość trasy pojazdu z towarem ma bardzo duże znaczenie – przekłada to się na realne koszty dla przedsiębiorstwa. Za każdy dodatkowo przejechany kilometr trzeba zapłacić, czy to bezpośrednio płacąc za paliwo czy wypłacając pensje kierowcy. Dlatego też wynikowa trasa dla jakiegoś problemu trasa powinna być możliwie jak najkrótsza. Z drugiej strony jednak parametry trasy często zmieniają się w czasie rzeczywistym i w takim przypadki wyniki musimy otrzymać w możliwie jak najkrótszym czasie. Jednak proces obliczania może być czasochłonny.

Obliczenia możemy podzielić na dwie fazy, pierwsza z nich to obliczanie macierzy odległości, kolejna to właściwy algorytm obliczania rozwiązania na podstawie wcześniej obliczonej macierzy. Które z zaimplementowanych algorytmów da jak najlepszy wynik w jak najlepszym czasie, oraz w jakich sytuacjach korzystać z nich? Z tego względy testowanie aplikacji podzielono na dwie części. Dla każdej części przeprowadzono szereg testów pokazujących, jaki wpływ na rozwiązanie ma wykorzystanie różnych sposobów obliczeń.

## Środowisko testowe

Wszystkie testy zostały uruchomione na komputerze o podanej specyfikacji:

* System operacyjny Windows 10 Education 64-bit (10.0, Build 16299)
* Procesor AMD Phenom II X4 955
* Pamięć 8192MB DDR3
* Płyta główna ASRock 880GM-LE

Wszystkie testy podobnie jak program zostały napisane w języku Groovy a następnie skompilowane. Na komputerze została zainstalowana Java SE 8 Update 91. Podczas wykonywania testów na komputerze nie było uruchomione inne oprogramowanie. Wykonywane były tylko testy. Do obliczania czasu wykonania programu wykorzystano obiekt klasy Date. Przykład pokazano poniżej.

Date czasStart = new Date()

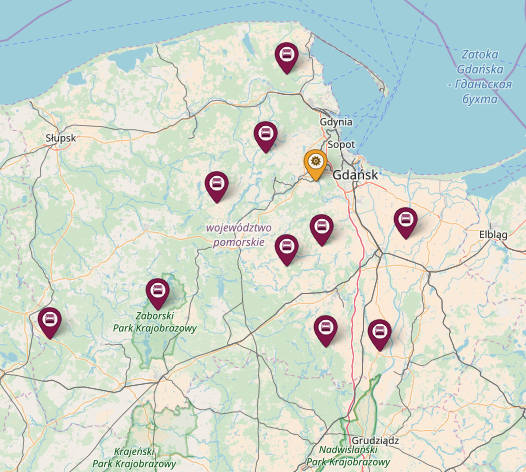
//Obliczenia

Date czasStop = new Date()

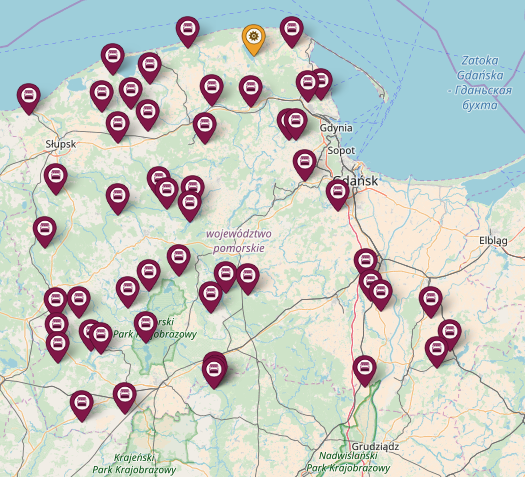
TimeDuration czasTrwania = TimeCategory.minus(timeStop, timeStart)

## Plan testów

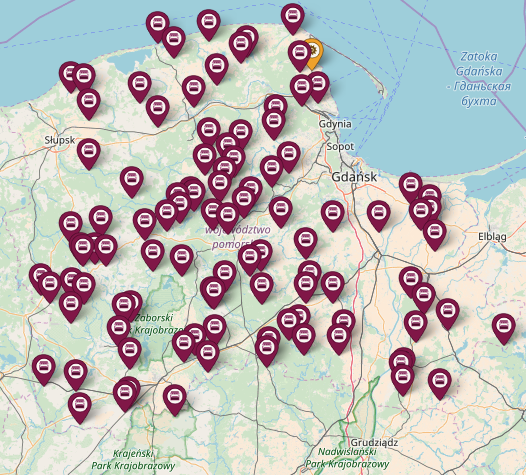
Przed rozpoczęciem testów zostały wygenerowane dane testowe. Dane testowe podzielone zostały na 20 kategorii. Ilość odbiorców wynosił kolejno 10, 20, 30, 40 etc. Dla każdej kategorii wylosowano 10 testów. Położenie każdego z wygenerowanych odbiorców jak i magazyn zostało wylosowane na powierzchni województwa pomorskiego. Po wylosowaniu położenia sprawdzono czy jest możliwe dotarcie do tego miejsca samochodem, dzięki czemu możliwe będzie późniejsze obliczenie odległości drogowej od i z tego punktu do pozostałych. Każdemu odbiorcy była przypisywana losowa wymagana ilość towaru. Wartość ta mieściła się w przedziale 5 a 15 jednostek. Maksymalna ładowność pojazdu była ustawiona na 200 jednostek. Spowodowało to, że wygenerowane trasy średnio posiadały po 20 przystanków. Liczba pojazdów nie była ograniczona. Przykładowe instancje testowe są pokazane kolejno na Rys. 28 o wielkości 10, Rys 29 o rozmiarze 50, Rys 30 100 oraz Rys 31 o rozmiarze 200.

****

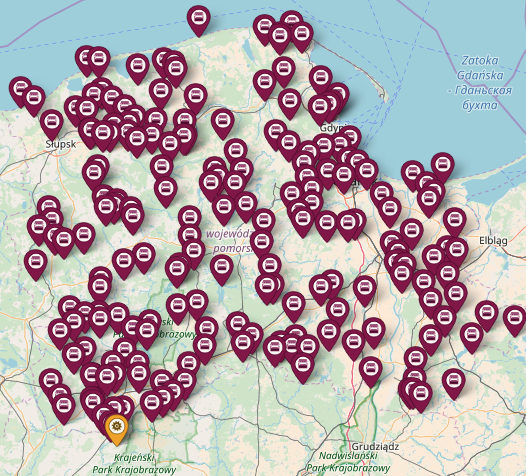
Rys. 28 Problem o rozmiarze 10

****

Rys. 29 Problem o rozmiarze 50

****

Rys. 30 Problem o rozmiarze 100

****

Rys. 31 Problem o rozmiarze 200

Testy wykonano w podanej kolejności. Na początku wczytano test, obliczono dla niego macierz odległości, jako odległość w linii prostej. Macierz ta została wykorzystana do obliczania kolejno wybranymi algorytmami oraz bibliotekami.

Wykorzystano kolejno:

* Algorytm oszczędności w wersji sekwencyjnej
* Algorytm najbliższego sąsiada
* Bibliotekę Jsprit z domyślnymi ustawieniami
* Bibliotekę Google Optimization Tools z domyślnymi ustawieniami

Dla których ustalono czas potrzebny do obliczeń. Rozwiązanie przekazano do obliczenia długości tras.

Następnie obliczono na nowo macierz odległości, tym razem obliczano odległości drogowe przy użyciu biblioteki GraphHopper, sposób jej działania oraz wykorzystane algorytmy opisane są w pierwszym rozdziale. Ponownie zmierzono czas obliczeń. Przy pomocy obliczonej macierzy wykorzystano te same algorytmy jak w poprzednim przykładzie mierząc czas oraz wynik rozwiązania.

## Wyniki testów

Na początku warto porównać czas wykonywania obliczeń macierzy odległości. Test wykonano na 10 osobnych testach dla każdej wielkości a później wynik ten uśredniono. Czas obliczeń macierzy dla odległości w linii prostej pokazane jest na wykresie przedstawionym na Rys 32.

Rys. 32 Wykres czasu obliczania macierzy dla odległości w linii prostej

Do obliczeń wykorzystano metodę „Hervesine”, została ona opisana w pierwszym rozdziale, ma ona złożoność liniową. Dla każdego odbiorcy musimy obliczyć odległość do magazynu oraz dla każdego innego odbiorcy. Daje nam to złożoność gdzie *n* to ilość odbiorców w problemie. Dla problemu z 10 odbiorcami czas wynosił natomiast dla problemy z 200 odbiorcami było to .

Na kolejnym wykresie (Rys. 33) przedstawiony został czas obliczania macierzy dla odległości drogowej.

Rys. 33 Czas obliczania macierzy dla odległości drogowej

Jak można zauważyć wielkości te znacznie się różnią od poprzednich wyników. Tutaj dla problemu o wielkości 10 czas obliczania wynosił aż . Natomiast dla problemu o wielkości 200 było to około 1 minuta oraz 47 sekund. Podobnie jak w poprzednim przypadku tutaj też trzeba było obliczyć odległości od jednego odbiorcy do pozostałych oraz do magazynu. Jednak w tym przypadku powstała macierz nie jest symetryczna. Spowodowane jest to tym, że odległość do i z punktu może być zupełnie inna, na przykład poprzez drogi jednokierunkowe. Do obliczania została wykorzystana mapa województwa pomorskiego pobrana ze strony [*http://download.geofabrik.de/europe/poland/pomorskie.html*](http://download.geofabrik.de/europe/poland/pomorskie.html)*.* Natomiast do analizy mapy oraz obliczenia najkrótszej odległości pomiędzy dwoma punktami została wykorzystana biblioteka GraphHopper.

Każdy z testów został także obliczony za pomocą wyżej wymienionych algorytmów. Pierwszym z nich jest wersja równoległa algorytmu oszczędności (C&W Savings). Kolejnym jest algorytm najbliższego sąsiada (ang. *nearest neighbours*, NN). Z drugiej strony zostały wykorzystane gotowe już biblioteki. Jsprit, która korzysta z metaheurystycznej metody Tabu Search. Uruchomiona została z domyślnymi ustawieniami. Limit iteracji został ustawiony na 2000. Jako ostatnia została wykorzystana biblioteka Google Optimization Tools (Google OT)*.* Podobnie jak poprzednia biblioteka – została wykorzystana z domyślnymi ustawieniami. W rozdziale pierwszym dokładniej zostały opisane domyślne ustawienia oraz algorytmy wykorzystywane przez tę bibliotekę. Na wykresach poniżej (Rys. 34, 35, 36, 37) przedstawiono otrzymane wyniki dla problemu z 10, 50, 100 oraz 200 odbiorcami.

Rys. 34 Wyniki dla 10 odbiorców

Rys. 35 Wyniki dla 50 odbiorców

Rys. 36 Wyniki dla 100 odbiorców

Rys. 37 Wyniki dla 200 odbiorców

Jak widać algorytm najbliższego sąsiada spisał się najgorzej dla każdej wielkości. Są to różnice znaczne – około 20% dla każdego z testów. Nie jest to zaskoczeniem, jest on bardzo prosty w działaniu, istnieją takie badania, które pokazują, że może on nawet zwrócić najgorsze możliwe rozwiązania [14]. Dla problemów z 10 odbiorcami pozostałe sposoby poradziły sobie podobnie. Algorytm oszczędności w kilku przypadkach wypadł nieco słabiej niż biblioteki. Sytuacja bardzo ciekawie wygląda dla testów problemów z ilościami odbiorców 100 oraz 200. Widać tutaj, że na najlepsze wyniki otrzymywaliśmy wykorzystując bibliotekę Jsprit. Za nią podobnie wypadają wyniki otrzymane przy użyciu biblioteki GoogleOT oraz algorytmu oszczędności. Różnice pomiędzy tymi dwoma sposobami jednak nie były tak duże, w niektórych przypadkach lepiej wypadał algorytm oszczędności, w innych natomiast wykorzystana biblioteka.

Powyższe wykresy przedstawiały obliczone wartości dla macierzy zbudowanej z bezpośrednich odległości. Inne wyniki otrzymano poprzez użycie macierzy obliczonej na podstawie odległości drogowych. Zostało to przedstawione na wykresach (Rys. 38, 39, 40, 41). Podobnie jak w poprzednim przypadku tutaj zostały pokazane wyniki dla problemów z 10, 50, 100 oraz 200 odbiorcami.

Rys. 38 Wynik dla 10 odbiorców

Rys. 39 Wynik dla 50 odbiorców

Rys. 40 Wyniki dla 100 odbiorców

Rys. 41 Wyniki dla 200 odbiorców

Wygenerowane wykresy przedstawiają się prawie identycznie do poprzednich – generowanych na podstawie macierzy odległości w linii prostej. Tutaj też algorytm najbliższego sąsiada okazał się być dużo gorszy od pozostałych, natomiast wykorzystanie biblioteki Jsprit dało najlepsze rezultaty.

Na podstawie tych danych można obliczyć czy oraz jaki wpływ na rozwiązanie końcowe ma wykorzystanie poszczególnych macierzy. Na tabeli poniżej (Tabela 4) przedstawiono o ile procentowo lepszy lub gorszy był wynik obliczany na podstawie odległości drogowej nad tym obliczanym odległości w linii prostej.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Savings** | **NN** | **Jsprit** | **Google OT** |
| **10** | 2.40 % | - 1.32 % | 0.18 % | 1.99% |
| **50** | 2.80 % | 2.95 % | 3.75 % | 2.78 % |
| **100** | 2.11 % | - 1.15 % | 4.09 % | 3.34 % |
| **200** | 2.58 % | 1.16 % | 5.07 % | 6.42 % |

Tabela 4 Różnica pomiędzy wynikami

Jak można zauważyć, że najmniejsza różnica pomiędzy wynikami jest na algorytmu NN. W tym wypadku też często (około 40% przypadków) okazywało się, że użycie macierzy odległości w linii prostej dawało lepszy wynik. Najlepszymi wynikami mogą pochwalić się biblioteki Jsprit oraz *Google OT*. Wielkość problemu też miała wpływ na różnicę, ta rosła wraz ze wzrostem rozmiaru problemu. Dla 200 odbiorców było to aż 6% różnicy, gdzie przy 50 odbiorcach wielkości były na poziomie 3%. Dla algorytmu oszczędnościnie ma wpływu ilość odbiorców, tutaj różnica utrzymywała się na poziomie 2.5%.

Dla każdego z wykorzystanych algorytmów został obliczony także czas wykonania. Na czas obliczeń nie ma większego wpływu typ użytej macierzy odległości. Dlatego też zostaną pokazane czasy dla obliczenia wyniku dla macierzy odległości w linii prostej. Jako pierwszy zostanie przedstawiony algorytm oszczędności. Wyniki można zobaczyć na Rys. 42.

Rys. 42 Czas obliczeń dla algorytmu C&W Savings

Dla wielkości problemów z 10 odbiorcami czas obliczeń wynosił około 11.4 ms, natomiast dla problemu z 200 odbiorcami czas ten wynosił już około 400 ms. Złożoność obliczeniowa testowanego algorytmu wynosi gdzie musimy utworzyć listę wszystkich możliwych oszczędności pomiędzy każdą parą punktów.

Kolejnym algorytmem, dla którego został zmierzony czas jest najbliższego sąsiada. Wyniki zostały przedstawione na Rys. 43.

Rys. 43 Czas obliczeń dla algorytmu NN

Czas wykonania algorytmu był bardzo niski, nawet dla problemu z 200 odbiorcami. Obliczenia w tym wypadku zostały wykonane w 98ms. Implementacja tego algorytmu polegała na wybraniu najmniejszej wartości w rzędzie w macierzy, daje to nam kwadratową złożoność obliczeniową. Szybkość obliczeń niestety nie idzie w parze z dobrymi wynikami. Te były najsłabsze z wszystkich o około 20%.

Rys. 44 Czas obliczeń z wykorzystaniem biblioteki Jsprit

W przypadku wykorzystania biblioteki Jsprit czas obliczeń wygląda zupełnie inaczej. Średni czas wykonania poszczególnych wielkości problemów można zobaczyć na wykresie na Rys. 44. Czas wykonania dla poszczególnych wielkości problemu jest znacznie wyższy. Już dla 100 odbiorców było to ponad 30 sekund, a dla 200 odbiorców są to prawie 2 minuty. Czas obliczeń w tym przypadku też rósł najdynamiczniej. Długi czas oczekiwania na wynik końcowy obliczeń idzie jednak w tym przypadku w parze z dobrym wynikiem. Biblioteka Jspit z domyślnymi ustawieniami w teście dawała najlepsze wyniki. Czas obliczeń można byłoby zredukować poprzez zmianę na przykład ilości iteracji – jednak miałoby to wpływ, na jakość otrzymanego wyniku. W ramach testów wykonano również test porównujący jak ilość iteracji ma wpływ na czas oraz wynik obliczeń. Do tego celu wykorzystano instancje z ilością 100 odbiorców. Ilość iteracji ustawiono po kolei na 10, 100, 500 oraz 1000 iteracji. Wyniki przedstawiono w poniższej tabeli (Tabela 5).

|  |  |  |
| --- | --- | --- |
| **Ilość iteracji** | **Średni czas obliczeń [ms]** | **Średnia różnica w długości trasy [%]** |
| 2000 | 30047 | 0% |
| 1000 | 22318 | 4.2% |
| 100 | 10651 | 5.3% |
| 10 | 459 | 16.3% |

Tabela 5 Różnice pomiędzy ilością iteracji

Jak można było się spodziewać ilość iteracji znacznie wpływa na wynik oraz czas obliczeń. Jeśli chodzi i czas trwania nie jest to drastyczna zmiana, przy ograniczeniu o połowę ilość iteracji (z 2000 na 1000), średnio czas spadł tylko o 8 sekund. W tym też przypadku uzyskaliśmy długość trasy wyższą o około 4.2%. Nawet dla 10 iteracji czas był duży, obliczenia trwały około 0.5 sekundy a uzyskaliśmy wynik gorszy od tego, który uzyskaliśmy wykorzystując algorytm oszczędności. Tam natomiast czas obliczeń było to jedynie 30ms. Wybór ilości iteracji zależy od tego jak szybko potrzebny jest nam wynik. Najoptymalniejszą opcją wydaje się wybranie domyślnej opcji – 2000 iteracji. Daje ona nam zdecydowanie najlepszy wynik, a czas wykonania nie jest drastycznie dłuższy porównując z 1000 iteracji. Na tak długi czas wykonania nawet dla 10 iteracji też w pewnym stopniu może mieć wpływ tłumaczenie danych z modelu wykorzystywanego w bibliotece na ten wykorzystywany w implementowanej w ramach pracy aplikacji.

Jako ostatnia testowana była druga z wybranych bibliotek - Google Optimization Tools. Średni czas obliczeń można zobaczyć na Rys. 45.

Rys. 45 Czas obliczeń z wykorzystaniem biblioteki Google OT

Maksymalny czas obliczeń dla tej biblioteki wynosił średnio 2.5 sekundy dla problemu z 200 odbiorcami. Wzrost nie był tutaj tak dynamiczny jak w przypadku drugiej testowanej biblioteki. Wyniki działania biblioteki porównywane były z wykorzystaniem algorytmu oszczędności*.* Jednak, jeśli chodzi o czas obliczeń te wyniki już nie były tak podobne. Dużo dłużej trwały obliczenia wykonywane przez bibliotekę. Dla porównania do obliczenia rozwiązania dla 200 odbiorców biblioteka potrzebowała około 2.5 sekundy, gdzie algorytm oszczędnościpotrzebował 0.4 sekundy. Wpływ na tak wysoki wynik najprawdopodobniej ma też translacja z modelu używanego w aplikacji na model i metody używane w bibliotece. Podobnie z wynikiem, po zakończeniu obliczeń trzeba z przetłumaczyć wynik na model wykorzystywany w aplikacji.

## Porównanie

Mając przed sobą zestaw wyników możemy ocenić, kiedy i w jakim przypadku najlepiej będzie wybrać określoną metodę obliczania macierzy odległości oraz wybranie zaimplementowanego algorytmu lub wykorzystanie biblioteki. Na początek warto przeanalizować algorytm najbliższego sąsiada. Wyniki uzyskane poprzez obliczenia były dalekie od ideału, mylił się w porównaniu z pozostałymi sposobami o około 20%. Jest to bardzo dużo. Dla przykładu weźmy pewien problem, w którym musieliśmy przebyć trasę o długości około 5000km. Przy wykorzystaniu wyżej wymienionego algorytmu ta trasa wydłużyłaby się o około 1000km. Biorąc pod uwagę, że samochód jedzie ze średnią prędkością około 70km/h potrzeba dodatkowo około 14 godzin, aby wykonać wszystkie dostawy. Przekłada się to na dużo większe średnie zużycie paliwa. Jak widać generuje to bardzo duże koszty transportu. Z drugiej jednak strony obliczenia wykonane przez ten algorytm były bardzo szybkie. Przeliczenie wszystkich tras na nowo dla 200 odbiorców wykona w około 80ms. Niestety uzyskany wynik dyskwalifikuje ten sposób obliczeń.

Kolejnym testowanym algorytmem jest Savings. W jego przypadku było znacznie lepiej, wyniki były porównywane z tymi, które otrzymaliśmy za pomocą biblioteki udostępnionej przez firmę Google oraz uruchomionej z domyślnymi ustawieniami. Czas obliczeń w tym przypadku bardzo dobrze wypadł na tle konkurencji. Dla obliczenie całkowitego rozwiązania dla problemu z 200 odbiorcami ten algorytm potrzebował jedynie około 0.4 sekundy. Dzięki czemu stawia go w gronie faworytów.

Z drugiej strony mamy dwie wybrane biblioteki. Pierwsza z nich to biblioteka Jsprit, dugą natomiast Google Optimization Tools. Wyniki tej pierwszej były bardzo dobre porównując je z pozostałymi testowanymi sposobami. Dla problemów z ponad 100 odbiorcami nie miała sobie równych. Jednak czas wykonania dla niej był bardzo długi. Biblioteka daje możliwość sterowaniem ilością iteracji, po której zatrzyma obliczenia. Zmniejszenie ich bazowej ilości o połowę dało dużo gorszy wynik, gdy czas obliczeń mimo wszystko pozostawał względnie bardzo duży porównując z pozostałymi bibliotekami.

Wyniki drugiej wykorzystywanej biblioteki nie były tak imponujące jak pierwszej biblioteki. Jednak czas wykonywania się był dużo krótszy. Wyniki były porównywalne z użyciem tylko i wyłącznie algorytmu oszczędności.

Warto zaznaczyć, że wybrane algorytmy (oszczędności oraz najbliższego sąsiada) w formie, w której zostały zaimplementowane wspierają tylko wariant CVRP, natomiast możliwości wykorzystania bibliotek oraz ich konfiguracji są znacznie większe. Najszerszymi możliwościami może pochwalić się biblioteka Google OT, która nie tylko wspiera poszczególne warianty VRP, ale też pomaga w rozwiązywaniu innych optymalizacyjnych problemów. Podczas tworzenia modelu do obliczeń w przypadku biblioteki Google OT możemy dodać wiele różnych ograniczeń. Dodatkowymi atutami wykorzystania wyżej wymienionej biblioteki są ustawienia. Możemy ustawić, jaki maksymalny czas możemy przeznaczyć na czas obliczeń. Co w dużym stopniu pomaga, gdy musimy otrzymać wynik w konkretnym czasie. Dodatkowo możemy sami wybierać z listy zaimplementowanych algorytmów, jakie chcemy wykorzystać. Fakt, że ta biblioteka jest aktywnie wspierana, rozwijana oraz wykorzystywana przez Google jest bardzo dużym atutem. Firma ta wykorzystuje ją między innymi do tworzenia swojej innej usługi – Google Street View, która polega na robieniu zdjęć podczas jechaniu samochodem. W tym momencie w samej Polsce tą funkcjonalnością pokryte są niemal wszystkie trasy. Sugeruje to, że biblioteka ta jest bardzo dobrze przetestowana oraz można jej zaufać. Niestety biblioteka ta bazowo została zaimplementowana w języku C++, jednak możliwe jest jej uruchomienie na platformie Java. Możliwe jest to dzięki specjalnie przygotowanym bibliotekom ładującym oraz uruchamiającym skompilowany bazowy kod. Niestety niesie to za sobą pewne konsekwencje, przez które niektóre funkcje konfiguracji nie są dostępne oraz samo uruchomienie obliczenia może powodować niekoniecznie łatwe do zidentyfikowania oraz naprawienia błędy. Natomiast druga testowana biblioteka tak bogatej funkcjonalności nie proponowała. W jej ramach został zaimplementowany tylko jeden algorytm, którego parametry możemy dowolnie modyfikować. Wspiera on także szereg różnych wariantów marszrutyzacji. Jest nadal rozwijany, niestety nie w tak dynamicznym tempie jak druga testowana aplikacja. Jej dodatkowym atutem jednak jest to, że jest napisana całkowicie w języku Java, przez to nie było żadnych problemów z integracją z tworzoną aplikacją.

Oprócz sposobu obliczeń na ostateczny wynik miała wpływ macierz odległości. Tutaj porównywane były dwie możliwości, odległość w linii prostej oraz odległość drogowa. Ta druga okazała się lepsza w większości przypadków. Największy wpływ miała ona na dla obliczeń z wykorzystaniem bibliotek, gdzie różnica wynosiła niekiedy nawet ponad 5%.

Niestety nie ma jednoznacznego wyniku, jaki sposób obliczeń wykorzystywać. Jednak, jeżeli możemy wygenerować wynik dostatecznie wcześnie to warto poczekać chwilę i wykorzystać bibliotekę Jsprit wraz z obliczaniem macierzy drogowej. Dla problemu z 200 odbiorcami wynik został otrzymany w ponad 2 minuty. Każdy kilometr krótsza trasa to dodatkowe oszczędności.

# Podsumowanie

Cel pracy, jakim było utworzenie systemu internetowego wspierającego marszrutyzację pojazdów został spełniony. Utworzona aplikacja może zostać uruchamiana w przeglądarce internetowej. Została zaimplementowana przy użyciu platformę programistyczną Angular. Wpierana jest ona przez serwis internetowy udostępniający usługi obliczania rozwiązania zadanego problemu. Serwis ten także został zaimplementowany w ramach pracy. Utworzona aplikacja pokrywa wszystkie założone wcześniej funkcjonalności.

Schemat obliczania rozwiązania można podzielić na dwa kluczowe elementy. Pierwszym z nich jest obliczanie odległości pomiędzy punktami. W systemie dodano do wyboru także dwie możliwości obliczania tras pomiędzy punktami. Może być to odległość w linii prostej lub też odległość drogowa, która obliczana jest na podstawie danych pobranych z mapy. Drugim elementem jest budowanie rozwiązania.

Zaimplementowano dwa algorytmy pozwalające na obliczanie rozwiązania. Były nimi: algorytm oszczędnościowego łączenia tras oraz algorytm najbliższego sąsiada. Dodatkowo do projektu dodano oraz uruchomiono dwie biblioteki. Pierwszą z nich była biblioteka Jsprit, dugą natomiast biblioteka Google Optimization Tools.

Przeprowadzono szereg testów porównujących możliwości wszystkich zaimplementowanych sposobów obliczania rozwiązania. Porównano dodatkowo, jaki ma wpływ ma typ macierzy odległości na końcowy wynik. Nie uzyskano jednoznacznego wyniku, który sposób jest najlepszy.

Utworzony system w pełni może mieć realny wpływ na ograniczenie kosztów w firmie przewozowej poprzez ograniczenie ilości przejechanych kilometrów oraz czasu potrzebnego na obsługę wszystkich odbiorców.

# Wykaz literatury

|  |  |
| --- | --- |
| [1] | G. Zimon, „Analiza kosztów w przedsiębiorstwach transportu samochodowego,” [Online]. Available: http://www.wneiz.pl/nauka\_wneiz/frfu/77-2015/FRFU-77-349.pdf. [Data uzyskania dostępu: 22 05 2018]. |
| [2] | G. Dantzig i J. Ramser, w *The truck dispatching problem.*, Manag. Sci., 1959, p. 80–91. |
| [3] | G. Laporte, „The Vehicle Routing Problem: An overview of exact and approximate algorithms,” 1991. |
| [4] | S. Basu, „Tabu Search Implementation on Traveling Salesman Problem and Its Variations: A Literature Survey,” *American Journal of Operations Research,* pp. 163-173, 2012. |
| [5] | „Statystyki OpenStreetMap,” [Online]. Available: https://www.openstreetmap.org/stats/data\_stats.html. [Data uzyskania dostępu: 22 05 2015]. |
| [6] | J. S. Gerhard Schrimpf, „Record Breaking Optimization Results Using the Ruin and Recreate Principle,” *Journal of Computational Physics,* p. 139–171, 2000. |
| [7] | „Capacitated Vehicle Routing Problem Library,” [Online]. Available: http://vrp.atd-lab.inf.puc-rio.br/index.php/en/. [Data uzyskania dostępu: 22 05 2018]. |
| [8] | „Single Page Application,” [Online]. Available: https://en.wikipedia.org/wiki/Single-page\_application. [Data uzyskania dostępu: 22 05 2018]. |
| [9] | „Angular,” [Online]. Available: https://angular.io. [Data uzyskania dostępu: 22 05 2018]. |
| [10] | „Leaflet,” [Online]. Available: https://leafletjs.com/. |
| [11] | „XMLHttpRequest specification,” [Online]. Available: https://www.w3.org/TR/XMLHttpRequest. [Data uzyskania dostępu: 22 05 2018]. |
| [12] | „Base64,” [Online]. Available: https://kb.iu.edu/d/aepm. [Data uzyskania dostępu: 22 05 2018]. |
| [13] | „WebSocket Protocol,” [Online]. Available: https://tools.ietf.org/html/rfc6455. [Data uzyskania dostępu: 22 05 2018]. |
| [14] | A. Y. A. Z. Gregory Gutina, „Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP,” *Discrete Applied Mathematics 117,* p. 81–86, 2002. |
| [15] | „OpenStreetMap,” [Online]. Available: https://www.openstreetmap.org. [Data uzyskania dostępu: 22 05 2018]. |

Wykaz tabel

[Tabela 1 Macierz odległości 12](#_Toc515820912)

[Tabela 2 Kolejne kroki algorytmu 12](#_Toc515820913)

[Tabela 3 Porównanie narzutu danych dla różnych sposobów komunikacji 39](#_Toc515820914)

[Tabela 4 Różnica pomiędzy wynikami 49](#_Toc515820915)

[Tabela 5 Różnice pomiędzy ilością iteracji 52](#_Toc515820916)

Wykaz rysunków

[Rys. 1 Przykład CVRP 8](#_Toc515820867)

[Rys. 2 Obliczanie oszczędności 9](#_Toc515820868)

[Rys. 3 Podział na grupy 9](#_Toc515820869)

[Rys. 4 Zrzut ekranu systemu logvrp.com 14](#_Toc515820870)

[Rys. 5 Diagram przypadków użycia 16](#_Toc515820871)

[Rys. 6 Diagram ERD pokazujący model danych 19](#_Toc515820872)

[Rys. 7 Planowana architektura systemu 20](#_Toc515820873)

[Rys. 8 Planowany zarys aplikacji klienckiej 21](#_Toc515820874)

[Rys. 9 Diagram sekwencji przedstawiający proces obliczania rozwiązania 22](#_Toc515820875)

[Rys. 10 Diagram przedstawiający architekturę aplikacji 24](#_Toc515820876)

[Rys. 11 Ekran główny aplikacji 25](#_Toc515820877)

[Rys. 12 Diagram klas 26](#_Toc515820878)

[Rys. 13 Ekran logowania aplikacji 27](#_Toc515820879)

[Rys. 14 Zakładka „Info” 28](#_Toc515820880)

[Rys. 15 Zakładka „Solutions” 29](#_Toc515820881)

[Rys. 16 Zakładka „Customers” 30](#_Toc515820882)

[Rys. 17 Zakładka „Problems” 30](#_Toc515820883)

[Rys. 18 Mapa z zaznaczonymi odbiorcami oraz magazynem 31](#_Toc515820884)

[Rys. 19 Siatka wraz z zaznaczonymi punktami 32](#_Toc515820885)

[Rys. 20 Ikona odbiorcy 32](#_Toc515820886)

[Rys. 21 Ikona magazynu 32](#_Toc515820887)

[Rys. 22 Trasa łącząca dwóch odbiorców oraz magazyn 33](#_Toc515820888)

[Rys. 23 Wyświetlone szczegóły trasy 33](#_Toc515820889)

[Rys. 24 Porównanie metod 35](#_Toc515820890)

[Rys. 25 Przeglądarki obsługujące XHR 36](#_Toc515820891)

[Rys. 26 Przeglądarki obsługujące Server-Sent Events 37](#_Toc515820892)

[Rys. 27 Przeglądarki obsługujące WebSocket 38](#_Toc515820893)

[Rys. 28 Problem o rozmiarze 10 41](#_Toc515820894)

[Rys. 29 Problem o rozmiarze 50 41](#_Toc515820895)

[Rys. 30 Problem o rozmiarze 100 42](#_Toc515820896)

[Rys. 31 Problem o rozmiarze 200 42](#_Toc515820897)

[Rys. 32 Wykres czasu obliczania macierzy dla odległości w linii prostej 43](#_Toc515820898)

[Rys. 33 Czas obliczania macierzy dla odległości drogowej 44](#_Toc515820899)

[Rys. 34 Wyniki dla 10 odbiorców 45](#_Toc515820900)

[Rys. 35 Wyniki dla 50 odbiorców 45](#_Toc515820901)

[Rys. 36 Wyniki dla 100 odbiorców 46](#_Toc515820902)

[Rys. 37 Wyniki dla 200 odbiorców 46](#_Toc515820903)

[Rys. 38 Wynik dla 10 odbiorców 47](#_Toc515820904)

[Rys. 39 Wynik dla 50 odbiorców 48](#_Toc515820905)

[Rys. 40 Wyniki dla 100 odbiorców 48](#_Toc515820906)

[Rys. 41 Wyniki dla 200 odbiorców 49](#_Toc515820907)

[Rys. 42 Czas obliczeń dla algorytmu C&W Savings 50](#_Toc515820908)

[Rys. 43 Czas obliczeń dla algorytmu NN 51](#_Toc515820909)

[Rys. 44 Czas obliczeń z wykorzystaniem biblioteki Jsprit 51](#_Toc515820910)

[Rys. 45 Czas obliczeń z wykorzystaniem biblioteki Google OT 53](#_Toc515820911)