Politechnika Krakowska
im. Tadeusza Kościuszki

Wydział Inżynierii Elektrycznej i Komputerowej

Aplikacja usprawniająca pracę kuriera – problem komiwojażera w praktyce

Aleksander Krzeszowski

Promotor:
dr inż. Damian Grela

Kraków, 15 lutego 2017

Spis treści

Wstęp

W literaturze można odnaleźć liczne prace skupiające się na analizie wydajności i optymalizacji różnych algorytmów rozwiązujących problem komiwojażera. Celem poniższej pracy jest stworzenie łatwej w obsłudze aplikacji, umożliwiającej odnalezienie optymalnej trasy łączącej wprowadzone miejsca docelowe. Założenie dostępności dla zwykłego użytkownika nakłada pewne wymagania na aplikację: interfejs musi być prosty w obsłudze, a aplikacja powinna być dostępna na różnych urządzeniach (komputery, tablety, urządzenia przenośne).

Wymagania te spełnia aplikacja internetowa – dostępna przez przeglądarkę. Taki rodzaj aplikacji pozwala na realizację architektury klient-serwer. W tym modelu obliczenia są wykonywane po stronie serwera, nie obciążając klienta, który wyświetla tylko interfejs aplikacji.

Kolejną konsekwencją przeznaczenia aplikacji dla zwykłych użytkowników jest konieczność zapewnienia odpowiedniej wydajności. Przetwarzanie danych powinno przebiegać w możliwie krótkim czasie, tak by użytkownik nie musiał czekać na zakończenie obliczeń. Znalezienie dokładnego rozwiązania problemu komiwojażera w czasie wielomianowym jest niemożliwe [4].

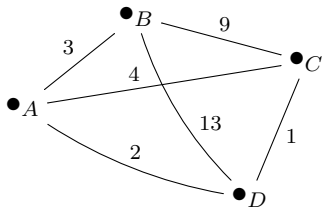
Poszukiwanie optymalnego rozwiązania już dla małej liczby miejsc docelowych wymagałoby znacznego czasu, nawet na wydajnym komputerze. Z tego powodu aplikacja powinna wykorzystywać algorytm sub-optymalny, który pozwala odnaleźć rozwiązanie w stosunkowo krótkim czasie, a odnalezione trasy są wystarczająco optymalne dla praktycznych zastosowań.

Rozdział 1

Definicja i rozwiązanie problemu

1.1 Określenie problemu

Problem komiwojażera polega na wyznaczeniu trasy łączącej wybrane punkty[1], przy dodatkowych warunkach: każdy punkt może zostać odwiedzony wyłącznie raz, poza wybranym punktem będącym początkiem i końcem trasy. Można więc powiedzieć, że rozwiązanie stanowi permutacja n punktów, a optymalnym rozwiązaniem jest permutacja o minimalnej sumie odległości między punktami[2].



Rysunek 1.1: Przykład symetryczny: Reprezentacja grafowa

[1]Pierwotnie problem dotyczył tras między miastami, przez co w opracowaniach lub algorytmach punkty pośrednie często nazywa się miastami

	A	B	C	D
A	0	3	4	2
B	3	0	9	13
C	4	9	0	1
D	2	13	1	0

Tabela 1.1: Przykład symetryczny: Macierz sąsiedztwa

Źródło: Opracowanie własne

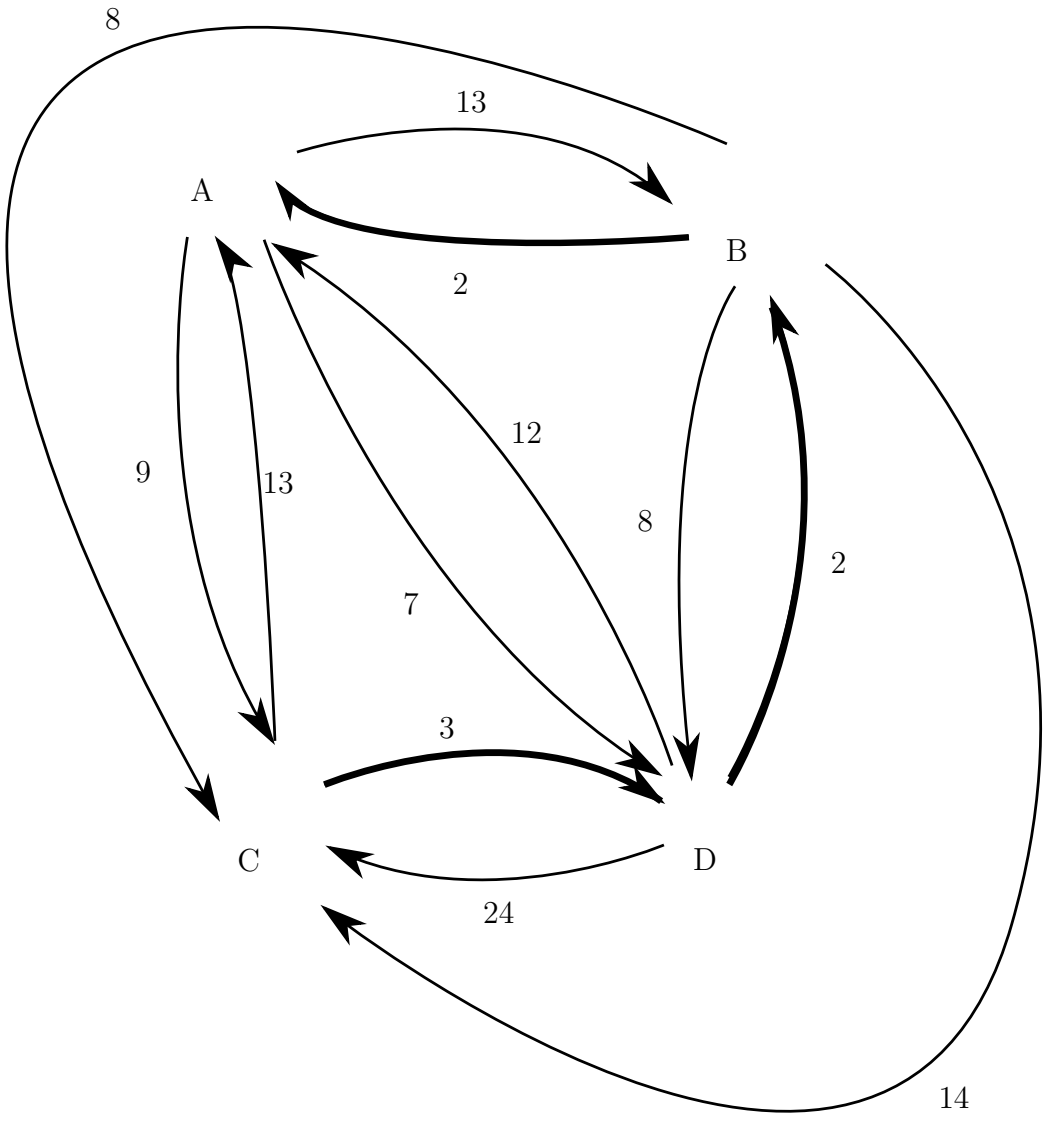
Przykładowy problem został przedstawiony na grafie 1.1. Przyjmując B za punkt startowy, optymalną trasą dla takiego zbioru punktów jest na przykład: $B \to A \to D \to C \to B$ o długości 15.

Punkty pośrednie stanowią wierzchołki grafu, a trasy je łączące to krawędzie o wagach równych odległościom między punktami. Powyższy prosty przykład to wariant symetryczny problemu komiwojażera – odległości między dwoma punktami są identyczne w każdym kierunku. Dla takiego grafu wystarczy odnaleźć wagi dla $\frac{n(n-1)}{2}$ krawędzi, ponieważ są nieskierowane.

Jednak w rzeczywistym zastosowaniu (a także w zrealizowanej aplikacji) mamy do czynienia z asymetryczną wersją problemu, a więc ze skierowanym grafem. Jest to spowodowane tym, że co prawda z dowolnego punktu możemy dotrzeć do innego, jednak trasy między dwoma punktami mogą być inne (na przykład ulice jednokierunkowe). W rezultacie konieczne jest odnalezienie $n^2 - n$ odległości między punktami.

Przykład problemu asymetrycznego znajduje się na grafie 1.2. Najkrótszą trasą rozpoczynającą się w punkcie C jest $C \to D \to B \to A \to C$ o długości 16.

Warto zauważyć że dla algorytmów nie ma znaczenia w jakiej jednostce jest wyrażona waga – można więc tym samym algorytmem optymalizować zarówno odległość, jak i czas przejazdu.



Rysunek 1.2: Przykład asymetryczny: Reprezentacja grafowa

	A	B	C	D
A	0	13	9	7
B	2	0	8	8
C	13	14	0	3
D	12	2	24	0

Tabela 1.2: Przykład symetryczny: Macierz sąsiedztwa

Źródło: Opracowanie własne

1.2 Algorytmy genetyczne

Ogólny przebieg algorytmu ewolucyjnego wygląda następująco:

Algorytm 1 Program ewolucyjny **Źródło**: [2]

```
1: procedure PROGRAM EWOLUCYJNY
2:     t ← 0
3:     ustal początkowe P(t)                    ▷ Inicjalizacja
4:     while not warunek zakończenia do
5:         t ← t + 1
6:         wybierz P(t) z P(t − 1)              ▷ Selekcja
7:         zmień P(t)                           ▷ Krzyżowanie i mutacja
8:         oceń P(t)                            ▷ Funkcja oceny
9:     end while
10: end procedure
```

W kolejnych podrozdziałach zostaną omówione poszczególne etapy programu.

1.2.1 Inicjalizacja

Inicjalizacja polega na wygenerowaniu populacji złożonej z losowych osobników. W przypadku problemu komiwojażera populację stanowi zbiór tras,

składających się z punktów pośrednich, dlatego należy spełnić warunek, że każdy element musi wystąpić dokładnie raz w wylosowanej trasie.

Najprostszym sposobem na spełnienie powyższego warunku jest przetasowanie zbioru wszystkich punktów. Opracowany program korzysta ze współczesnej wersji algorytmu Fishera-Yatesa[1]. Ma on złożoność $O(n)$ względem oryginalnego algorytmu o złożoności $O(n^2)$. Algorytm polega na wykonaniu n zamian między n-tym elementem zbioru, a losowym elementem, gdzie n jest równe liczbie elementów zbioru.

Wybór większego rozmiaru populacji zapewnia większą różnorodność, a w rezultacie zwiększa prawdopodobieństwo odnalezienia lepszego rozwiązania. Jednak zwiększanie tego parametru wydłuża czas działania algorytmu, co może być niepożądane przez użytkownika. Dlatego rozmiar populacji jest jednym z parametrów dostępnych do modyfikacji przez użytkownika – może on samodzielnie dobrać rozmiar odpowiadający wymaganej poprawności i czasowi przetwarzania.

1.2.2 Funkcja oceny

Ocena rozwiązania polega na przypisaniu wartości liczbowej danemu rozwiązaniu. Dla trasy jest to jej całkowita długość, czyli suma długości odcinków od wybranego punktu, między wszystkimi punktami pośrednimi i od ostatniego punktu pośredniego do punktu początkowego.

Punkty są przechowywane w aplikacji w postaci grafowej, więc odnalezienie długości trasy jest proste. Wszystkie punkty zawiera zbiór odległości do każdego innego punktu. Odległości są przechowywane we standardowej kolekcji `Dictionary<T>`, która jest zaimplementowana w postaci hash mapy, umożliwiającej odnalezienie elementu w czasie $O(1)$. Dlatego obliczanie całkowitej odległości ma złożoność liniową.

Aplikacja umożliwia także wyszukiwanie tras bez określonego punktu początkowego. Przy tej samej liczbie wszystkich punktów skorzystanie z tej opcji zwiększa liczbę możliwych tras, co utrudnia odnalezienie tej najkrótszej.

Ponieważ kurierzy po dostarczeniu wszystkich przesyłek zazwyczaj nie muszą wracać do sortowni, dodana została również opcja wyłączenia powrotu do punktu początkowego.

1.2.3 Selekcja

1.2.4 Krzyżowanie

1.2.5 Mutacja

Rozdział 2

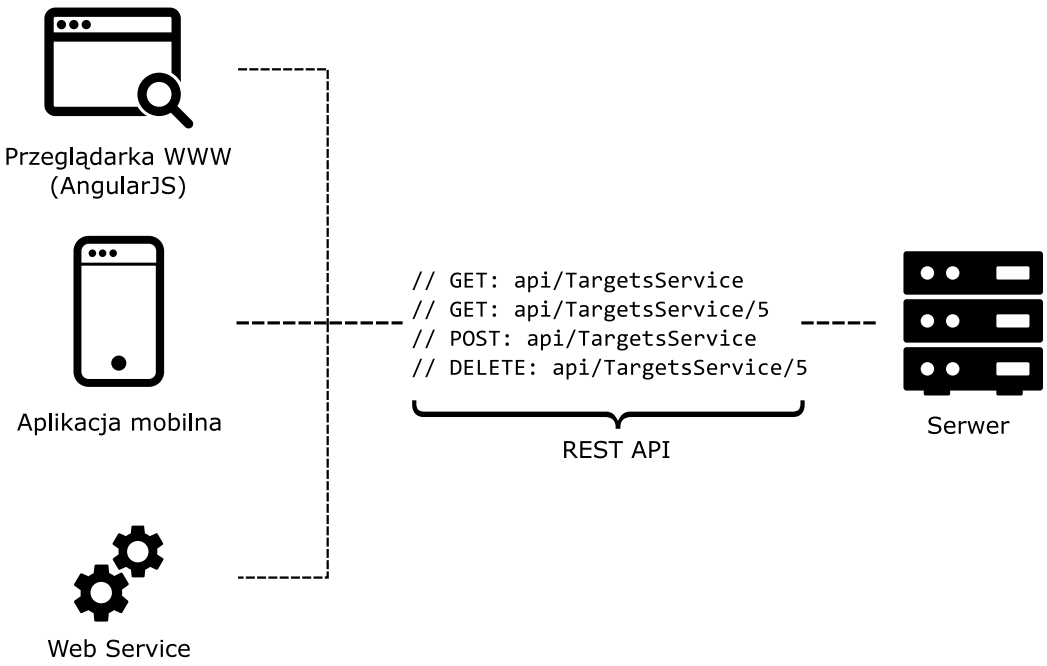
Implementacja

2.1 Struktura projektu

(Opis najważniejszych projektów i klas, architektura aplikacji, schematy. Wzorce projektowe w osobnym podrozdziale? MVVM, Factory, IoC,) Aplikacja została zrealizowana w języku C#. Jest to zorientowany obiektowo język korzystający z .NET Framework [3]. Do tworzenia aplikacji w tej technologii można skorzystać z darmowego środowiska Visual Studio. Język ten został wybrany głównie ze względu na możliwość stworzenia w nim aplikacji internetowej w technologii ASP.NET oraz dobrą wydajność pozwalającą na szybkie dokonanie obliczeń.

C# jest kompilowany do kodu pośredniego (CIL), przez co jego szybkość jest nieco niższa od języków kompilowanych do języka maszynowego, jak np. C, C++. Jednak skompilowany kod pośredni jest mocno zoptymalizowany, co pozwala na użycie go także w czasochłonnych obliczeniach, a niższa wydajność względem kodu natywnego jest praktycznie niezauważalna.

Praca w Visual Studio polega na stworzeniu *solucji*, czyli zbioru powiązanych projektów tworzących po skompilowaniu gotową aplikację. Podział na projekty umożliwia skorzystanie z różnych języków i kompilatorów w jednej solucji, a także logiczny podział komponentów. Powiązania między projektami są określane przez *referencje*. Po określeniu które projekty są zależne od



Rysunek 2.1: Schemat komunikacji między serwerem aplikacji oraz potencjalnymi klientami usługi sieciowej

Źródło: Opracowanie własne

innych, kompilator potrafi automatycznie ustalić kolejność ich budowania. Solucja opisywanej aplikacji składa się z trzech projektów:

TSP

Przeglądarkowy interfejs użytkownika, napisany w AngularJS, który korzysta z API[1] stworzonego w ASP.NET. API zostało zrealizowane jako *webservice* w konwencji REST[2]: udostępnia wszystkie funkcjonalności przy użyciu standardowych metod HTTP, co pozwala łatwo napisać alternatywny interfejs, na przykład w formie aplikacji mobilnej. Schemat komunikacji z API został przedstawiony na rysunku 2.1. Klient może pobrać listę punktów pośrednich na mapie, wysyłając żądanie typu `GET`, lub podając identyfikator punktu wybrać pojedynczy punkt. Zarówno odpowiedzi serwisu jak i żąda-

[1]Application Programming Interface

[2]Representional State Transfer

nia klienta składają się z obiektów zserializowanych do tekstowego formatu JSON[3], który stanowi alternatywę dla XML. Pobierając wybrany punkt z serwisu, odpowiedź zostanie zwrócona w poniższym formacie:

```
{
    "Id":"fff9a6bc-d36e-4c30-a49b-aa7022bfa352",
    "Name":"Basztowa 1, 33-332 Kraków, Polska",
    "Location":
    {
        "Latitude":50.066285384750863,
        "Longitude":19.935379028320312
    }
}
```

Dodawanie punktów jest realizowane przez żądanie `POST`, a usuwanie przez `DELETE`. Próba pobrania lub usunięcia nieistniejącego punktu spowoduje zwrócenie standardowego błędu HTTP 404.

Aby umożliwić jednoznaczną identyfikację punktów, każdy punkt przy dodawaniu otrzymuje indywidualny, losowy identyfikator GUID[4], który wygląda następująco: `5B7665CB-0B67-4D14-85F6-CDE6C5ACA7C8`. Składa się on z 32 znaków heksadecymalnych, przez co prawdopodobieństwo wylosowania identycznego identyfikatora jest znikome.

W zrealizowanej aplikacji z API serwisu korzysta strona napisana w HTML oraz JavaScript z biblioteką AngularJS. Dzięki skorzystaniu z możliwości JavaScript, strona nie musi być przeładowywana przy wykonywaniu żądania, przez co przypomina natywną aplikację okienkową pod względem wygody i szybkości obsługi.

Solver

Silnik odnajdujący optymalne trasy na podstawie zbioru punktów. Utworzenie silnika w osobnym projekcie sprawia że jest kompilowany do osobnego

[3]JavaScript Object Notation

[4]Globally Unique Identifier

pliku DLL. Jest on zupełnie niezależny od interfejsu użytkownika (nie posiada referencji do projektu TSP). Umożliwia to skorzystanie z jego możliwości w innych aplikacjach, oraz implementację alternatywnych interfejsów, np. w formie zwykłej aplikacji okienkowej na system Windows.

Opis wykorzystanych algorytmów znajduje się w rozdziale 1.

Solver.Tests

Testy jednostkowe, sprawdzające poprawność działania silnika. Zostały napisane z pomocą otwartoźródłowej biblioteki NUnit, umożliwiającej tworzenie testów jednostkowych w .NET Framework. Najważniejsze testy zostaną opisane w rozdziale 3.2.

2.2 Integracja systemów zewnętrznych

(Opis API, sposobu integracji i napotkanych problemów)

2.2.1 Google Maps

2.2.2 OSRM: Open Source Routing Machine

Rozdział 3

Testowanie aplikacji

3.1 Testy manualne

(Prezentacja działania aplikacji (screenshoty), uruchomienie dla dużej liczby punktów, przy różnej konfiguracji)

3.2 Testy jednostkowe

(Opis testów jednostkowych sprawdzających poprawność komponentów)

3.3 Badanie wydajności

(Porównanie poprawności i szybkości rozwiązania dla różnych konfiguracji algorytmu)

Podsumowanie

(Podsumowanie pracy, potencjalne możliwości rozwoju aplikacji.) Język C# umożliwia integrację z kodem C++, co pozwala na uzyskanie jeszcze lepszej wydajności, przy zachowaniu pełnej funkcjonalności po przepisaniu części obliczeniowej na język kompilowany do kodu natywnego. Dzięki oparciu API o standard HTTP możliwe jest łatwe stworzenie dodatkowych interfejsów użytkownika, w formie aplikacji mobilnej lub okienkowej. Elastyczna konstrukcja silnika, zbudowanego z użyciem wzorców projektowych pozwala na dodanie i rozbudowę jego poszczególnych części, takich jak algorytmy krzyżowania i selekcji lub całkowitą zamianę typu algorytmu na inny niż genetyczny.

Bibliografia

[1] Richard Durstenfeld. Algorithm 235: Random permutation. *ACM*, 7(7):420–, July 1964.

[2] Zbigniew Michalewicz. *Algorytmy genetyczne + struktury danych = programy ewolucyjne*. Wydaw. Naukowo-Techniczne, Warszawa, 2003.

[3] Microsoft. MSDN: C#. `https://msdn.microsoft.com/pl-pl/library/kx37x362.aspx`. [dostęp 9.02.2017].

[4] Christos H. Papadimitriou. The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, 4(3):237, 1977.