JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 10 (2019) 18-23
Wysłane: 2018-11-14
Przyjęte: 2018-11-20

Modyfikacje algorytmów planowania trasy uwzględniające ograniczenia czasowe i odległościowe

Mateusz Wolanin*, Klaudia Korniszuk, Jakub Smołka
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Artykuł przedstawia modyfikacje algorytmów wyszukiwania ścieżki w grafie mające na celu wprowadzenie ograniczeń: czasowych lub odległościowych do znalezionej trasy. Zmodyfikowane zostały dwa algorytmy: A* oraz BFS. Zaproponowana została również modyfikacja algorytmu A*, która łączy atuty tych dwóch algorytmów – wygenerowanie najkrótszych tras o jak najmniejszej liczbie wierzchołków. Zmodyfikowane algorytmy umożliwią stworzenie aplikacji pozwalającej na łatwiejsze i bardziej oszczędne poruszanie się z wykorzystaniem usług typu rowerem miejski.

Słowa kluczowe: wyznaczanie trasy; rower miejski; algorytm A*; algorytm BFS

*Autor do korespondencji.

Adresy e-mail: mat.wol20@outlook.com, klaudia714@gmail.com, jakub.smolka@pollub.pl

Modification of path-finding algorithms introducing time and distance limitations

Mateusz Wolanin*, Klaudia Korniszuk, Jakub Smołka
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. This paper describes modifications of path-finding algorithms. The modifications add time and distance constraints to generated paths. A* and BFS algorithms are modified. Additionally, A* algorithm modification which combines the advantages (generating the shortest routes with the smallest number of vertices) of A* and BFS is presented.. This allows for creating a route planning app that enables users of bike sharing services to travel more easily and economically.

Keywords: route planning; bike sharing system; algorithm A*; algorithm BFS

*Corresponding author.

E-mail addresses: mat.wol20@outlook.com, klaudia714@gmail.com, jakub.smolka@pollub.pl

1. Wstęp

Podróżowanie zawsze było istotne dla człowieka. Niezależnie czy podróż jest krótka lub długa, największym problemem jest wyznaczenie tak trasy, aby w optymalnym czasie i koszcie dotrzeć w docelowe miejsce. W dzisiejszych czasach dostępnych jest wiele środków transportu: samochody, samolotów lub rowery. Ten artykuł skupia się na planowaniu trasy dla tej ostatniej opcji podróży.

Istnieją systemy wypożyczalni rowerów (z ang. BSS – bike sharing system), gdzie użytkownik może wypożyczyć z jednej stacji rower, by oddać go w dowolnej innej stacji. Dzięki takiemu rozwiązaniu użytkownik może poruszać się po mieście szybciej i sprawniej, niż jakby to robił pieszo.

Zazwyczaj firma udostępniająca rowery miejskie posiada następujący model biznesowy: z góry określony czas od wypożyczenia jest darmowy. Wypożyczenie roweru na dłużej niż wyżej wymieniony czas kosztuje określoną w cenniku ilość gotówki. Dzięki takiemu rozwiązaniu istnieje możliwość przejechania od stacji do stacji bez wydania pieniędzy na wypożyczenie roweru. Takie rozwiązanie ma zapewne na celu zachęcenie ludzi do korzystania z ich środka transportu.

Celem badań autorów było stworzenie takich algorytmów, które umożliwią podróżowanie wyżej wymienionym środkiem transportu w określonym darmowym czasie. Można to wykonać wyznaczając pomiędzy stacją początkową, a końcową stacje pośrednie, na których można wymienić rower. Dzięki temu czas się resetuje i ponownie użytkownik ma do dyspozycji darmowy czas na przejazd do następnej stacji. Przy sprawnym nawigowaniu można bezproblemowo przejechać na miejskim rowerze prawie całe miasto.

W tym artykule zostaną przedstawione algorytmy umożliwiające wyznaczenie drogi od stacji początkowej do stacji końcowej poprzez stacje pośrednie. W tym celu zostaną uwzględnione ograniczenia czasowe lub odległościowe. Zaproponowane algorytmy zostaną porównane ze względu na liczbę wyznaczonych przystanków, koszt trasy oraz czas działania.

2. Przegląd istniejących rozwiązań

Podczas wyszukiwania istniejących rozwiązań nie znaleziono artykułów poruszających ten problem. Badania o zbliżonej tematyce zostały przestawione poniżej,

W pierwszej pracy autorzy publikacji [1] opisują problem rozmieszczenia rowerów na stacjach tak, aby użytkownicy

mogli wypożyczyć rower z wybranej, najczęściej najbliższej stacji. W tej pracy zostały zaproponowane dwa algorytmy wyznaczające trasę z od punktu startowego do najbliższej stacji początkowej gdzie znajdują się rowery, następnie od stacji początkowej do stacji końcowej z wolnymi miejscami dokującymi oraz od stacji końcowej do punktu końcowego. Algorytmy dla *m* stacji oraz *n* użytkowników wyznaczały dla każdego użytkownika trasę. Według autorów tego artykułu, im mniejszy czas podróży, tym większe z niej zadowolenie (jakość podróży). Dlatego ten współczynnik stał się podstawową miarą do wyznaczenia trasy przez algorytm. Pierwszy z nich o nazwie GTP (Greedy Trip Planning) automatycznie przypisuje do użytkownika trasę, która w zbiorze wycieczek ma aktualnie najwyższą jakość pasującą do wymagań użytkownika. Przy tych założeniach można wywnioskować, że tylko pierwsi użytkownicy rowerów miejskich będą mieli zapewnioną najwyższą jakość trasy. Następni po nich użytkownicy będą musieli się liczyć z tym, że na niektórych stacjach startowych nie ma już możliwości wypożyczenia roweru, natomiast na stacjach końcowych może zabraknąć miejsca do zostawienia nowoprzybyłych rowerów. Drugim zaproponowanym algorytmem jest HTP (Humble Trip Planning). W przeciwieństwie do poprzedniego algorytmu GTP, HTP najpierw eliminuje konflikty wśród użytkowników poprzez przypisywanie tras o najwyższej jakości do użytkowników, które mają najwyższy współczynnik zmiany jakości podróży po zmianie przystanku początkowego i\lub końcowego. Dopiero po przydzieleniu użytkownikom najbardziej problematycznych tras reszta użytkowników otrzymuje najlepsze, pod względem jakości, trasy.

Kolejny artykuł [2] opisuje wyznaczenie trasy od punktu początkowego, przez stację początkową i końcową roweru miejskiego, do punktu końcowego dla jednego użytkownika. Jako ograniczenia w wyznaczaniu powyższej trasy przyjęto czas podróży lub odległość do pokonania, bezpieczeństwo na drodze oraz poziom zanieczyszczenia powietrza. Dopiero po uwzględnieniu tych trzech warunków obliczają całkowity koszt podróży na wybranych odcinkach na mapie i na podstawie tego algorytm Dijkstry ma za zadanie wyznaczyć końcową trasę rowerzysty. Na potrzeby przetestowania algorytmu autorzy publikacji [2] stworzyli testowe miasto które zawiera prawie 700 wierzchołków oraz 2620 łuków. W ten sposób odtworzono typowe środowisko miejskie z blokami, drogami oraz parkami. Zaprezentowany algorytm bada wszystkie możliwe trasy pod kątem długości, zanieczyszczenia powietrza oraz bezpieczeństwa i na tej podstawie wyznacza trasę rowerową dla użytkownika. Oprócz tego algorytm również zwraca alternatywne trasy różniące się między sobą tymi trzema współczynnikami.

Następny artykuł [3] przedstawia rozwiązanie problemu podobnego do przedstawionego w tym artykule - wyznaczyć trasę pomiędzy punktami, tak aby była najkrótsza. Różnicą jest to, że w tym rozwiązaniu wyznaczono trasę pomiędzy atrakcjami turystycznymi w danym mieście. Rozwiązują oni problem znalezienia trasy pomiędzy atrakcjami turystycznymi tak, aby wygenerowana droga była najkrótsza oraz aby przedstawiała różnorodne atrakcje turystyczne. W tym celu wprowadzili oni miarę *typeS*. Przyjmuje ona coraz to niższe wartości, gdy kolejna atrakcja jest podobna do poprzedniej. Jako algorytm zastosowali oni algorytm kolonii mrówek (ant colony alghoritm). Algorytm stworzyli Dorigo i Gambardella na podstawie obserwacji kolonii mrówek szukającej najkrótszej trasy od mrowiska do pożywienia. [4] Polega on na tym, że każda mrówka zostawia po sobie ślad tzw. feromon. Gdy mrówka znajdzie pożywienie, wraca do mrowiska tą samą drogą wzmacniając ślad pozostawiony w drodze do pożywienia. Kolejne mrówki wyczuwają ten ślad i kierują się tam gdzie prowadzi. Jako że feromon ulatnia się wraz z powietrzem w czasie, to długie ścieżki lub te, które nie nigdzie nie prowadzą zostają zapomniane. W ten sposób mrówki znajdują najkrótszą ścieżkę do pożywienia nie komunikując się ze sobą tylko pozostawiając ślady.

Ostatni artykuł [5] opisuje problem wyszukiwania trasy w dużym mieście bądź kraju. Według autorów publikacji [5] algorytmy Dijkstry oraz A* działają dobrze w przypadkach, kiedy mapa, na której działają jest względnie niewielka. Natomiast w przypadku wyszukiwania ścieżki pomiędzy dwoma punktami w dużym mieście lub w kraju, z powodów ograniczonej pamięci urządzenia lub ograniczonych zasobów procesora te dwa algorytmy mogą zwracać wyniki w zbyt długim czasie. By rozwiązać ten problem w artykule [5] zaproponowano użycie algorytmu Hierarchical A* (HAS), który jak sama nazwa wskazuje, dodaje mechanizm hierarchii do tradycyjnego algorytmu A*. Algorytm ten dzieli mapę na części, a w miejscu przecięcia krawędzi grafu z granicami sektora wstawia punkty. W przypadku gdy algorytm ma wyznaczyć trasę pomiędzy punktami w różnych sektorach, to najpierw wyznacza trasę z wierzchołka początkowego do wierzchołka końcowego przez punkty znajdujące się na krawędziach sektorów. Dopiero po tym kroku jest generowana trasa pomiędzy wierzchołkami, a punktami na granicach sektorów. Dzięki takiemu rozwiązaniu algorytm może działać na ograniczonej liczbie wierzchołków, generując trasę tylko dla jednej części mapy na raz. Według testów przeprowadzonych przez autorów opisywanej pracy, wraz ze wzrostem odległości pomiędzy punktem startowym, a końcowym od pewnego momentu HAS zaczął zwracać wyniki szybciej niż algorytm A*.

3. Opis badań

Zbadane zostaną trzy algorytmy, które zostały zmodyfikowane, aby uwzględniały zadane ograniczenia czasowe lub odległościowe:

- BFS – jeden z najprostszych algorytmów, służy za podstawę do bardziej złożonych algorytmów [6]. Autorzy zmodyfikowali ten algorytm, dodając uwzględnianie wag krawędzi oraz selekcję wierzchołków na podstawie zadanego ograniczenia. Schemat tej modyfikacji przedstawiony jest na poniższym przykładzie 1.

Przykład 1. Pseudokod algorytmu BFS uwzględniającego zadane ograniczenia

```
Wybierz punkt początkowy i końcowy
Ustal wartość ograniczenia
Dodaj punkt początkowy do kolejki
Dopóki kolejka nie jest pusta
    Pobierz i usuń pierwszy element E z kolejki
    Ustaw E jako odwiedzony
```

```
    Jeśli E jest punktem końcowym
        Zwróć wygenerowaną trasę
    Koniec warunku
    Dla każdego nieodwiedzonego sąsiada S elementu E, dla których waga krawędzi N – S jest mniejsza niż zadane ograniczenie
        Dodaj S do kolejki
    Koniec pętli
Koniec pętli
Zwróć Null
```

- A* - jeden z najlepszych algorytmów pod względem efektywności wyszukiwania najkrótszej ścieżki w grafie [7,8]. Wykorzystuje on funkcję kosztu, składającej się z sumy kosztu dotarcia do danego wierzchołka oraz przewidywanego kosztu dotarcia z danego punktu do wierzchołka końcowego. Podobnie jak w przypadku algorytmu BFS, ten algorytm również został zmodyfikowany, aby uwzględniał zadane ograniczenia czasowe lub odległościowe. Schemat działania algorytmu A* jest przedstawiony na poniższym przykładzie 2.

Przykład 2. Pseudokod algorytmu A* uwzględniającego zadane ograniczenia

```
Wybierz punkt początkowy i końcowy
Ustal wartość ograniczenia
Dodaj punkt początkowy do kolejki
Ustaw funkcję kosztu = 0 dla punktu początkowego
    Dopóki kolejka nie jest pusta
    Pobierz i usuń element E z kolejki, który posiada najmniejszą funkcję kosztu
    Ustaw E jako odwiedzony
    Jeśli E jest punktem końcowym
        Zwróć wygenerowaną trasę
    Koniec warunku
    Dla każdego nieodwiedzonego sąsiada S elementu E, dla których waga krawędzi N – S jest mniejsza niż zadane ograniczenie
        Oblicz P koszt dotarcia z punktu początkowego do S
        Oblicz K przewidywany koszt dotarcia z S do punktu końcowego
        Zapisz funkcję kosztu P+K dla S
    Koniec pętli
Koniec pętli
Zwróć Null
```

- Mod A* - zmodyfikowany przez autorów algorytm A*, który wraz z uwzględnianiem zadanego ograniczenia, oblicza minimalną liczbę stacji pośrednich oraz minimalny koszt pomiędzy stacjami. Na podstawie tych danych algorytm ogranicza liczbę przesiadek na danej trasie, poprzez wybieranie stacji pośrednich do których koszt dojazdu jest większy niż obliczony minimalny koszt.

  Minimalną odległość między stacjami jest obliczana następującym wzorem:

$$MD = d - n \times OD \tag{1}$$

  gdzie: d – odległość pomiędzy stacją aktualną, a stacją końcową, n – minimalna liczba stacji pośrednich, OD – wartość zadanego ograniczenia.

Schemat działania algorytmu jest przedstawiony na poniższym *przykładzie 3*.

Przykład 3. Pseudokod modyfikacji algorytmu A*

```
Wybierz punkt początkowy i końcowy
Ustal wartość ograniczenia
Dodaj punkt początkowy do kolejki
Oblicz minimalną liczbę przesiadek MP, dzieląc koszt trasy z punktu początkowego do końcowego przez ograniczenie
Ustaw funkcję kosztu = 0 dla punktu początkowego
Dopóki kolejka nie jest pusta
    Jeśli kolejka zawiera stację końcową
        Pobierz i usuń stację końcową E z kolejki
    W przeciwnym wypadku
        Pobierz i usuń element E z kolejki, który posiada najmniejszą funkcję kosztu
    Koniec warunku
    Ustaw E jako odwiedzony
    Jeśli E jest punktem końcowym
        Zwróć wygenerowaną trasę
    Koniec warunku
    Oblicz liczbę stacji potrzebną do dotarcia do E
    Jeśli E jest większe lub równe MP
        Zwiększ MP o 1
        Wyczyść kolejkę
        Dodaj element początkowy do kolejki
        Ustaw funkcję kosztu = 0 dla punktu początkowego
        Rozpocznij następną iterację pętli
    W przeciwnym wypadku
        Oblicz minimalny dystans pomiędzy E, a stacją sąsiadującą MD za pomocą wzoru 1
    Koniec warunku
    Dla każdego nieodwiedzonego sąsiada S elementu E, dla których waga krawędzi N – S jest mniejsza niż zadane ograniczenie i większa niż MD
        Oblicz P koszt dotarcia z punktu początkowego do S
        Oblicz K przewidywany koszt dotarcia z S do punktu końcowego
        Zapisz funkcję kosztu P+K dla S
    Koniec pętli
    Jeśli istnieje nieodwiedzony sąsiad S elementu E, dla których waga krawędzi N – S jest mniejsza niż zadane ograniczenie oraz kolejka jest pusta
        Zwiększ MP o 1
        Wyczyść kolejkę
        Dodaj element początkowy do kolejki
        Ustaw funkcję kosztu = 0 dla punktu początkowego
    Koniec warunku
Koniec pętli
Zwróć Null
```

Algorytmy działały na testowych grafach ważonych. Zostały wygenerowane za pomocą żądań sieciowych do [9]. Budowane były na podstawie współrzędnych stacji początkowej oraz końcowej i zwracały plik JSON, zawierający maksymalnie trzy trasy. Z tych danych były wybierane te trasy, które zawierały jak najmniejszą odległość lub czas trwania podróży. W ten sposób zapewniono wagi dla każdego możliwego połączenia w grafie. Każdy testowy graf posiadał 86 wierzchołków.

Algorytmy wygenerowały 7310 tras. Są to wszystkie możliwe połączenia pomiędzy wierzchołkami w testowych grafach. Trasy te zostaną ocenione pod względem trzech kryteriów:

1) Liczba przesiadek na trasie
2) Koszt trasy pomiędzy stacją początkową, a końcową
3) Czas wyszukiwania trasy

Pierwszym głównym kryterium porównawczym algorytmów jest liczba przesiadek między stacjami

wymaganymi by dotrzeć do celu podróży. Trasa wygenerowana przez algorytmy ma być najmniej problematyczna dla użytkownika. Częste i niepotrzebne przesiadki mogą zniechęcić i sfrustrować rowerzystów korzystających z aplikacji, która ma zaimplementowany algorytm nieprzystosowany do tego typu zadania.

Drugim głównym kryterium porównawczym algorytmów jest długość wygenerowanej trasy. Koszt będzie mierzony w metrach w przypadku, gdy jako źródło danych będzie wykorzystany graf ważony z wagami krawędzi równymi dystansowi pomiędzy stacjami. Natomiast, gdy źródłem danych będzie graf, który jako wagi krawędzi będzie miał czas wymagany do przebycia odległości pomiędzy dwoma stacjami, koszt trasy będzie mierzony w sekundach.

Porównanie algorytmów na podstawie kosztu podróży pozwoli określić, który algorytm zwraca najkrótszą, czasowo lub odległościowo, trasę pomiędzy dwoma zadanymi punktami. Za długa trasa ponownie może zniechęcić użytkowników, którzy chcą przejechać od punktu A do punktu B w jak najkrótszym czasie.

Ostatnim, pobocznym kryterium porównawczym będzie średni czas wymagany wygenerowania jednej trasy. Pomimo tego, że komputery osobiste jak i telefony są coraz bardziej wydajne i generowanie trasy w grafie posiadającym 86 wierzchołków nie powinno stanowić wyzwania, autorzy chcą przedstawić jak ich modyfikacje wpływają na szybkość działania algorytmu. Kryterium to będzie badane na komputerze o następujących podzespołach:

- procesor: Intel i7 – 4710HQ, 2,5 GHz,
- 8 GB pamięci RAM,
- dysk SSD Samsung 860 EVO 500 GB.

W przypadku grafu z wagami krawędzi równymi odległościom pomiędzy stacjami wartość ograniczenia wynosi 5000 metrów. Jest to średnia odległość, którą rowerzysta przejedzie z prędkością 16 km/h w 20 minut [10]. Natomiast w przypadku grafu z wagami krawędzi równymi czasowi wymaganym do przejechania pomiędzy dwoma stacjami wartość ograniczenia wynosi 1200 sekund, czyli 20 minut.

4. Wyniki badań

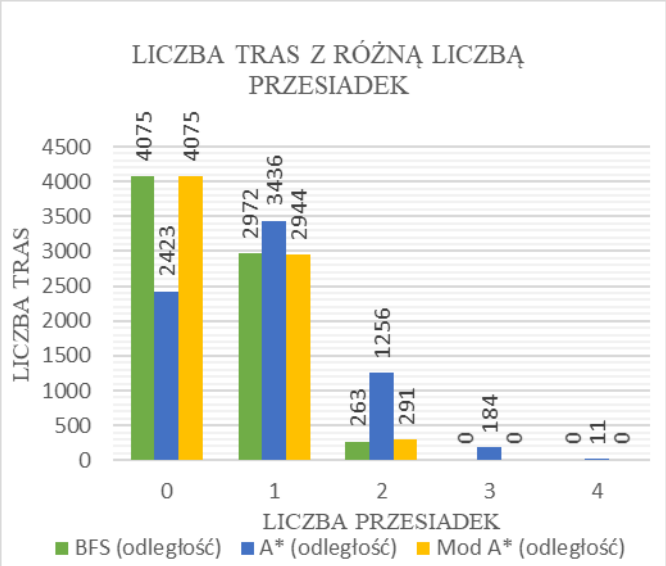
W tym rozdziale zostaną przedstawione porównania wyników opisanych w *rozdziale 3* algorytmów. Najpierw zostaną przedstawione wyniki dla algorytmów działających na grafie z wagami krawędzi odpowiadającymi odległości pomiędzy dwoma stacjami. Następnie zostaną zaprezentowane wyniki dla tych samych algorytmów, które działały na grafie z wagami krawędzi odpowiadającymi czasowi wymaganemu na pokonanie odległości pomiędzy dwoma stacjami.

4.1. Algorytmy uwzględniające ograniczenia odległościowe

Wykorzystane w tym podrozdziale algorytmy zostały zaprezentowane pod poniższymi nazwami:

- BFS (odległość), BFS (odl.) – algorytm BFS uwzględniający zadane ograniczenie,
- A* (odległość), A* (odl.) – algorytm A* uwzględniający zadane ograniczenie,
- Mod A* (odległość), Mod A* (odl.) – zmodyfikowany algorytm A*.

Te trzy algorytmy przyjmują jako ograniczenie odległość podaną w *rozdziale 3.*



Rys. 1. Porównanie liczby przesiadek – algorytmy uwzględniające ograniczenia odległościowe.

Tabela 1. Liczba tras, w danym przedziale kosztu, wygenerowanych przez algorytmy uwzględniające ograniczenie odległościowe.

	Liczba tras		
Koszt (metry)	**BFS (odl.)**	**A* (odl.)**	**Mod A* (odl.)**
<= 5000	4091	4292	4282
> 5000	3219	3018	3028
> 6250	2700	1830	1855
> 7500	1797	929	1006
> 8750	840	400	478
> 10000	248	148	154
Średni koszt trasy	5205,56	4700,58	4772,76

4.2. Algorytmy uwzględniające ograniczenia czasowe

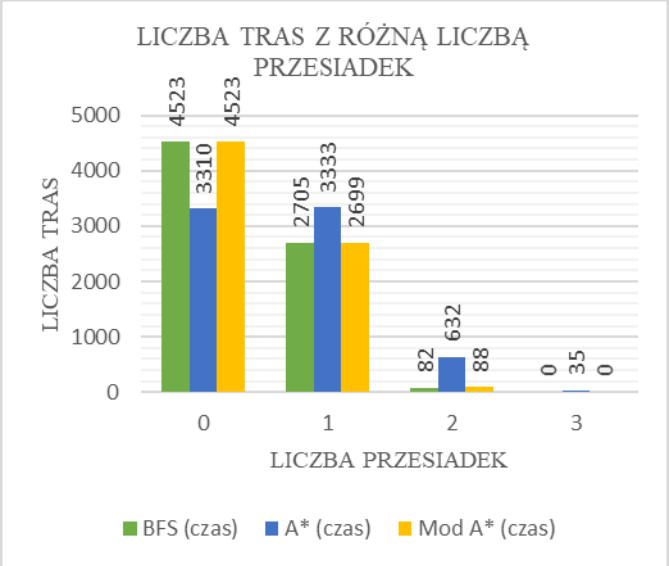
Wykorzystane w tym podrozdziale algorytmy zostały zaprezentowane pod poniższymi nazwami:

- BFS (czas) – algorytm BFS uwzględniający zadane ograniczenie,
- A* (czas) – algorytm A* uwzględniający zadane ograniczenie,
- Mod A* (czas) – zmodyfikowany algorytm A*.

Te trzy algorytmy przyjmują jako ograniczenie czas podany w *rozdziale 3.*

Tabela 2. Liczba tras, w danym przedziale kosztu, wygenerowanych przez algorytmy uwzględniające ograniczenie czasowe.

	Liczba tras		
Koszt (sekundy)	**BFS (czas)**	**A* (czas)**	**Mod A***
<= 1200	4531	4625	4622
> 1200	2779	2685	2688
> 1500	2254	1412	1421
> 1800	1378	602	628
> 2100	499	184	217
> 2400	77	38	40
Średni koszt trasy	1151,64	1055,76	1064,17



Rys. 2. Porównanie liczby przesiadek – algorytmy uwzględniające ograniczenia czasowe.

4.3. Czasy wygenerowania tras

Tabela 3. Czasy wygenerowania 7310 tras przez dany algorytm

Algorytm	Czas (ms)
BFS z zadanymi ogr.	297
A* z zadanymi ogr.	923
Modyfikacja A*	339

5. Wnioski

W przypadku algorytmów uwzględniających ograniczenia odległościowe, zarówno BFS jak i Mod A* wygenerowały podobną liczbę tras o zadanej liczbie stacji pośredniej. Ich wyniki różnią się w przypadku tras posiadających więcej niż jedną przesiadkę, na korzyść algorytmu BFS. Algorytm A* wygenerował najmniej tras nieposiadających przesiadek. Jako jedyny też wygenerował trasy prowadzące przez trzy lub cztery stacje pośrednie. Mogłoby się wydawać, że najkrótsza trasa powinna prowadzić bezpośrednią drogą od punktu początkowego do końcowego zawierająca ewentualne stacje pośrednie znajdujące się na tej drodze jednak algorytm A* zwrócił takie wyniki, ponieważ trasy wygenerowane przez [9], nie są zawsze możliwie najkrótsze. Spowodowane jest to algorytmem zwracającym trasy zaimplementowanym w [9]. Faworyzuje on ścieżki rowerowe i jeśli do punktu trasa może prowadzić przez ścieżkę rowerową, to zostanie ona zwrócona nawet w przypadku, gdy istnieją o wiele krótsze trasy.

Na podstawie wyników zawartych w tabeli 1, algorytm A* zwrócił najkrótsze trasy z pośród wszystkich algorytmów. Średni koszt w przypadku tego algorytmu wyniósł około 4701 metrów. Niewiele dłuższe trasy zwróciła modyfikacja A*, gdzie średnia długość tras wyniosła około 4773 metry. Porównując te dwa algorytmy na podstawie liczby tras w danym przedziale kosztu, to oba algorytmu posiadają zbliżone do siebie wyniki. Większa rozbieżność znajduje się w przypadku tras o koszcie większym niż 7500 oraz 8250 metrów. Algorytm BFS wygenerował najmniejszą liczbę tras o koszcie mniejszym lub równym 5000 metrów. Równocześnie w innych przedziałach, algorytm ten wygenerował największą liczbę tras.

W przypadku algorytmów uwzględniających ograniczenia czasowe, wyniki są podobne do tych wygenerowanych przez analogiczne algorytmy uwzględniające ograniczenia odległościowe. Tutaj też BFS oraz modyfikacja A* wygenerowały najmniejszą liczbę tras posiadających przynajmniej jedną przesiadkę. Natomiast różnica pomiędzy wynikami tych dwóch algorytmów jest znacząco mniejsza. Modyfikacja algorytmu A* wygenerowała o sześć więcej tras z dwoma przesiadkami niż BFS. A* wygenerował najgorsze wyniki z analogicznych powodów opisanych w przypadku algorytmów uwzględniających ograniczenia odległościowe.

Na podstawie wyników zawartych w tabeli 2, trasy o najmniejszym koszcie wygenerował algorytm A*, gdzie średni koszt trasy wyniósł 1056 sekund. Natomiast najdłuższe trasy zwrócił algorytm BFS. W tym przypadku średni koszt jest o prawie 100 sekund większy niż średni koszt algorytmu A*. Modyfikacja A* wygenerowała trasy o średnim koszcie równym 1064 sekundy, niewiele większym niż w przypadku algorytmu A*. Również, algorytmy te zwróciły podobną liczbę tras w poszczególnych przedziałach kosztu dla pojedynczej trasy. Algorytm BFS ponownie wygenerował najmniej tras o koszcie mniejszym lub równym 1200 sekund. Stworzył on również ponad dwa razy więcej tras posiadających koszt większy niż 1800 oraz 2100 sekund niż konkurencyjne algorytmy.

W tabeli 3 zawarto czasy wygenerowania wszystkich możliwych tras w grafach testowych przez poszczególne algorytmy. Najkrócej trasy tworzył algorytm BFS. Potrzebował na to mniej niż 300 milisekund. Zaproponowana przez autorów modyfikacja wygenerowała trasy w trzykrotnie mniejszym czasie niż podstawowa wersja algorytmu A*. Powodem tego jest mniejsza ilość badanych wierzchołków przy generowaniu pojedynczej trasy. Dzięki temu modyfikacja czasowo zbliżyła się do algorytmu BFS. A* potrzebował najwięcej czasu na wygenerowanie poszczególnych tras. Wyniósł on 923 milisekund.

Z pośród przetestowanych algorytmów najlepsza okazała się zaproponowana przez autorów modyfikacja A* (Mod A*). Generowała ona trasy o zbliżonej liczbie przesiadek tak jak w przypadku algorytmu BFS w krótkim czasie. Dodatkowo trasy posiadały niewiele większy koszt niż algorytm A*, który generował najkrótsze trasy, ale za to z największą liczbą przesiadek. Dodatkowo algorytm A* potrzebował najwięcej czasu do wygenerowania wszystkich tras spośród zaprezentowanych algorytmów

Literatura

[1] L GZhi Li, Jianhui Zhang, Jiayu Gan, Pengqian Lu, Fei Lin, Large-Scale Trip Planning for Bike-Sharing Systems, IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems, 2017

[2] Leonardo Caggiani, Rosalia Camporeale, Michele Ottomanelli, A real time multi-objective cyclists route choice model for a bike-sharing mobile application, Politecnico di Bari, 2017

[3] Wen Ouyang, Chang Wu Yu, Pei-Ju Huang, Huai-Tse Chang, Non-commutative path planning for tours with diversified attractions, Chung Hua University, 2017.

[4] Jing Luan, Zhong Yao, Futao Zhao, XinSong, A novel method to solve supplier selection problem: Hybrid algorithm of genetic algorithm and ant colony optimization, Mathematics and Computers in Simulation, Elsevier, 2019

[5] Haifeng Wang, Jiawei Zhou, Guifeng Zheng, Yun Liang, HAS:Hierarchical A-Star algorithm for big map navigation in special areas, 2014 International Conference on Digital Home, IEEE, 2014

[6] K. Khantanapoka, K. Chinnasarn: Pathfinding of 2D & 3D Game Real-Time Strategy with Depth Direction A*Algorithm for Multi-Layer, Eighth International Symposium on Natural Language Processing, IEEE, 2009

[7] W. Lu: Beginning Robotics Programming in Java with LEGO Mindstorms, Rozdział 9, 2016

[8] A. Chaudhari, M. Apsangi, A. Kudale: Improved A-star Algorithm with Least Turn for Robotic Rescue Operations, Computational Intelligence, Communications, and Business Analytics, Springer Nature, 2017

[9] Mapy Google, https://www.google.pl/maps/dir/ [dostęp 10.10.2018]

[10] dr inż. Tadeusz Kopta, mgr Aleksander Buczyński, Marcin Hyła, mgr inż. Bartłomiej Lustofin, Konkurencyjność roweru w zakresie czasu podróży, Warszawa-Kraków, czerwiec 2012