Politechnika Warszawska

Wydział Matematyki i Nauk Informacyjnych

Piotr Nagrodkiewicz

nr albumu: 177330

Systemy planowania zadań z uwzględnieniem ograniczeń czasowo – przestrzennych

Praca magisterska na kierunku INFORMATYKA

Praca wykonana pod kierunkiem

Prof. Marcina Paprzyckiego

Warszawa 2005

Spis treści:

Wykaz skrótów i akronimów wraz z rozwiązaniami

AI	–	Artificial Intelligence
FIPA	–	Foundation for Intelligent Physical Agents
GPS	–	Globar Positioning System
HTML	–	Hyper Text Mark-up Language
HTN	–	Hierarchical Task Network
IEEE	–	Institute for Electrical and Electronics Engineers
JESS	–	JAVA Expert System Shell
OWL	–	Web Ontology Language
RDF	–	Resource Description Framework
WWW	–	World Wide Web
XML	–	Extensible Mark-up Language

Wstęp

Niniejsza praca traktuje o systemach planowania zadań uwzględniających ograniczenia czasowo-przestrzenne. Będzie się ona koncentrowała na przykładzie systemu wspomagającego planowanie podróży. Zostanie podsumowany stan badań w tej dziedzinie - opisane zostaną różne scenariusze zastosowania jak i cele stawiane systemom wspomagającym podróżowanie, aby umożliwić dokładne poznanie warunków, w jakich przyjdzie im już w niedalekiej przyszłości pracować oraz ustalić, z jakimi innymi systemami będą one musiały współpracować, aby realizować cele stawiane przez użytkownika. Rzuci to światło na wymogi stawiane architekturze takich systemów. Niniejsza praca przedstawi również aktualną sytuację w tej dziedzinie - jest stawianych bardzo dużo postulatów co do funkcjonalności systemów planowania podróży, jednak stosunkowo niewiele prac zajmuje się stroną praktyczną takich systemów.

W części praktycznej niniejszej pracy zostanie zaproponowany algorytm umożliwiający zaplanowanie podróży między dwoma dowolnymi punktami wykorzystujący wiele różnych środków transportu. Będzie on stanowił krok do przodu w stosunku do aktualnie dostępnych systemów wspomagający podróżowanie, które bardzo często obsługują tylko jeden środek transportu. System przyszłości będzie musiał umożliwiać połączenie wielu środków transportu. Przygotowany algorytm będzie korzystał z danych zbieranych w ontologicznie zorientowany sposób przez inne moduły większego systemu wspomagającego podróżowanie, powstającego w ramach *Travel Support Project* [14]. Architektura przygotowanego rozwiązania będzie umożliwiała łatwą wymianę wykorzystywanych modułów, co będzie miało duże znaczenie w momencie, gdy pojawią się nowe, lepsze rozwiązania. Oprócz dostarczenia użytkownikowi planu podróży będzie on także umożliwiał personalizację przejawiającą się w automatycznej ocenie przygotowywanych propozycji. Jest to równie ważne jak samo planowanie – ten sam plan może być różnie oceniany przez dwie osoby o różnych preferencjach.

1 Przedstawienie zastosowań systemów planowania zadań uwzględniających ograniczenia czasowo-przestrzenne

1.1 Zastosowania ogólne

Systemy planowania zadań znajdują szerokie zastosowanie w dzisiejszym świecie. Stosuje się je wszędzie tam, gdzie człowiek miałby problemy ze sprostaniem rozmiarowi problemu lub gdzie po prostu zadania są monotonne i z powodzeniem mogą być wykonywane przez maszyny. Dlatego też często są wykorzystywane do sterowania robotami na halach produkcyjnych, gdzie decydują o ich ruchach[1], czy zarządzają dostarczaniem części do taśm produkcyjnych. Mogą też tworzyć plany wykonania urządzeń elektronicznych, gdzie poziom skomplikowania układów jest bardzo wysoki i trudno jest znaleźć najbardziej ekonomiczny proces ich wytwarzania[2]. Znajdują też zastosowanie we wsparciu logistycznym wielkich przedsiębiorstw - decydują co, gdzie i kiedy ma być przetransportowane [12, 18]. Mogą być też wykorzystywane do planowania procesów, na przykład, pozyskiwania informacji, gdzie decydują o kolejności zadawanych kwerend – ich zadaniem jest maksymalne skrócenie czasu oczekiwania na ostateczną odpowiedź oraz minimalizację ilości zadawanych zapytań pomocniczych (na przykład: rozproszone bazy danych). Systemy planowania funkcjonują również w świecie komputerów, gdzie są wykorzystywane do przydziału czasu procesora lub, na przykład, do routowania danych w sieciach transmisyjnych, czy też grach komputerowych. Znajdują także zastosowanie w sterowaniu pojazdami bezzałogowymi (na przykład w kosmosie - sonda Deep Space One - lub w wojsku).

[1] Aspekty teoretyczne i praktyczne można znaleźć w [19].

[2] Przykładem może być system Electro-Mechanical Design and Planning System wykorzystywany do produkcji odbiorników i transmiterów mikrofalowych [24].

Jednakże systemy planowania zadań to nie tylko rozwiązania stosowane w świecie techniki czy też wielkich korporacji. Coraz częściej wykorzystywane są one także do układania planów zajęć w szkołach i na uczelniach, czy też wspomagania planowania przebiegu projektów – komputery pomagają w ustaleniu terminów wykonania poszczególnych podzadań i przypisują do nich wykonawców (pracowników) posiadających odpowiednie kwalifikacje. Często są zdolne do brania pod uwagę dodatkowych ograniczeń, które mogą wpływać na przydział odpowiednich zasobów.

Wiedząc gdzie takie systemy znajdują zastosowanie, warto zadać sobie pytanie jak one to robią? Część problemów można rozwiązać w prosty sposób, po prostu sprawdzając wszystkie możliwości i wybierając najlepsze ze znalezionych rozwiązań (*brute force*). Część można rozwiązać w sposób analityczny (np. sterowanie ruchem ramienia robota). Z reguły jednak stawiane problemy są na tyle skomplikowane i złożone, że nie jest to możliwe. W takich przypadkach często ratunku szuka się w sztucznej inteligencji (*AI*) – algorytmach przeszukiwania drzew możliwości wykorzystujących heurystyki, sieciach neuronowych, czy też algorytmach ewolucyjnych. Ich słabością jest to, iż nie zawsze czas, jaki jest im potrzebny do rozwiązania problemu, jest akceptowalny, a czasami nawet trudny do przewidzenia. W takich przypadkach niekiedy stosuje się rozwiązania ze wsparciem człowieka, który naprowadza system na dobre rozwiązanie.

1.2 Transport

Systemy planowania znajdują także zastosowanie w systemach wspomagających transport. Głównie chodzi tu o planowanie sterowania ruchem w mieście przy pomocy sygnalizacji świetlnej, gdzie zadaniem jest tak kierować ruchem, aby zwiększyć ogólną przepustowość dróg, zmniejszyć czas oczekiwania podczas postoju na światłach, a przy okazji pośrednio ograniczać zużycie paliw i emisję szkodliwych substancji do atmosfery [25].

Prowadzone są także projekty badawcze mające na celu pomóc ludziom w podróżowaniu środkami transportu miejskiego. Można tu przytoczyć ciekawy pomysł wspomagania jazdy autobusami miejskimi w Dublinie [1]. System umożliwia dostęp do rozkładów jazdy, wskazuje jak dojść do najbliższych przystanków i jest w stanie zaplanować trasę dojazdu w dowolne miejsce w mieście. Potrafi także reagować na zdarzenia losowe takie, jak awarie autobusów. Oprócz tego umożliwia personalizację działania pod kątem upodobań użytkownika.

Z innych ciekawych pomysłów warto wspomnieć o idei zwiększenia wykorzystania możliwości transportu przy pomocy prywatnych samochodów [9]. Dillenburg, Wolfson oraz Nelson proponują rozwiązanie zakładające utrzymywanie bazy danych pojazdów, które poruszają się po mieście (wraz ze śledzeniem ich aktualnego położenia) i prowadzenie swoistej giełdy okazji biorąc pod uwagę takie czynniki jak cel trasy każdego pojazdu, ilość wolnych miejsc, czy też opłatę, jaką kierowca chce ewentualnie pobierać za podwiezieni innych osób. Sam pomysł może się wydawać trochę nierealny (de facto tworzy społeczne przedsiębiorstwo taksówkowe), ale w przyszłości, gdy cena paliwa będzie wysoka lub, gdy na drogach będzie zbyt wiele pojazdów, może okazać się rozwiązaniem godnym uwagi z braku lepszych alternatyw.

1.3 Planowanie podróży

Jak widać w dwóch poprzednich rozdziałach, celem systemów planujących zadania – jak zresztą całej informatyki – jest pomoc człowiekowi w jego codziennych problemach. Jednym z tych problemów jest planowanie podróży. Wiele osób spędza wiele godzin próbując znaleźć sposób dotarcia w odległe miejsce w szybki i w miarę możliwości tani sposób, a później organizując tam sobie atrakcyjny pobyt.

Najpierw trzeba odnaleźć sposób dotarcia na miejsce – to już samo w sobie nie jest proste ze względu na wyszukanie wszystkich dogodnych przesiadek.

Gdy ten problem jest rozwiązany, rodzi się kolejne pytanie - gdzie będzie się mieszkać? Jak spośród licznych hoteli wybrać właśnie ten, który będzie najbardziej odpowiedni? I jak zdobyć ich listę w miejscu, gdzie jeszcze nigdy się nie było? Każdy byłby zadowolony mając do dyspozycji pomoc elektronicznego agenta, który zrobiłby to za niego, najlepiej biorąc pod uwagę osobiste preferencje i upodobania. Stąd zrodził się pomysł stworzenia *Osobistego Agenta Podróży* (ang. *Personal Travel Agent*). Nazwa ta została zaczerpnięta z „*FIPA Personal Travel Assistance Specification*" [11]. FIPA (Foundation for Intelligent Physical Agents) to międzynarodowa organizacja standaryzacyjna IEEE (Institute of Electrical and Electronics Engineers) zajmująca się standardami i promowaniem technologii opartych na agentach.

Osobisty Agent Podróży powinien być w stanie wyszukać odpowiednie połączenia dla zadanego startu i celu podróży przy narzuconych ewentualnych ograniczeniach czasowych. Dotyczy to również drogi powrotnej. Idealnie by było, gdyby na czas pobytu był w stanie znaleźć odpowiedni hotel, i gdy jest to wyjazd turystyczny, zaproponować zwiedzanie muzeów, zabytków i innych atrakcji, oczywiście dobranych pod kątem preferencji, zainteresowań oraz możliwości finansowych użytkownika. Podczas odbywanie podróży agent powinien czuwać nad jej przebiegiem. Reagować na opóźnienia dokonując korekt planu, a także zadbać o to by miejsce w hotelu ciągle czekało, gdy nie ma już możliwości dotarcia na czas... Brzmi to trochę futurystycznie, ale już niedługo może stać się rzeczywistością. Największym problem leżącym na przeszkodzie realizacji tej koncepcji już dziś jest brak wysoko rozwiniętych środków zdobywania i przechowywania olbrzymich ilości bardzo różnorodnych informacji takich jak rozkłady jazdy, informacje o dostępności miejsc, informacji o hotelach, restauracjach, muzeach, itp. Prace nad rozwiązaniem tych problemów ciągle trwają. Odpowiednie źródła danych są już dostępne w Internecie. Jednak dane w nich dostępne nie posiadają jednolitego formatu, a dodatkowo w większości przypadków są one przedstawiane w formie czytelnej dla człowieka, a nie dla komputera. Co więcej, informacje te bywają ze sobą nawzajem sprzeczne. Mimo to, z każdym dniem zbliża się chwila, gdy w planowaniu podróży, zwłaszcza tych biznesowych, człowiek będzie wyręczany przez elektronicznych doradców.

2 Przedstawienie aktualnego stanu badań i zagadnień związanych z funkcjonowaniem *Osobistego Agenta Podróży*

2.1 Rozwiązania działające już dziś

Pewne rozwiązania informatyczne wspomagające planowanie podróży są dostępne już dziś. Niestety mają one ograniczone możliwości, albo działają w obrębie tylko jednego środka transportu. Jako przykład można tu podać serwis *Polskich Kolei Państwowych*[3], który udostępnia poprzez interfejs WWW rozkłady jazdy pociągów dla dowolnych stacji oraz udostępnia system wyszukiwania połączeń. Niestety działa on tylko dla pociągów i nie daje możliwości zaplanowania podróży w miejsce, gdzie nie da się dojechać koleją. Co więcej, użytkownik musi znać stację początkową i celową, aby móc wykonać przeszukiwanie. W praktyce sprowadza się to do konieczności znania najbliższej miejscowości ze stacją kolejową w okolicach startu i końca podróży. Jego niewątpliwą zaletą jest fakt, iż jego baza danych obejmuje połączenia kolejowe praktycznie z całej Europy. Serwisów tego typu jest oczywiście więcej.

Innym ciekawym przykładem już istniejących systemów są systemy nawigacji satelitarnej. Chyba najpopularniejszym rozwiązaniem tego typu, przynajmniej w Polsce, jest *Automapa*[4]. Każdy, kto posiada odpowiednie oprogramowanie, odbiornik GPS oraz palmtopa może w każdej chwili zorientować się gdzie jest. System jest w stanie podać położenie użytkownika z dokładnością do paru metrów. Może również zaplanować trasę dla samochodu między dowolnymi dwoma punktami na mapie i w czasie jazdy informować kierowcę pojazdu o przebiegu trasy. Jest to realizowane zarówno poprzez graficzną ilustrację aktualnego położenia na mapie wraz z zaznaczoną trasą

[3] Polskie Koleje Państwowe. Dostępne w World Wide Web: http://www.pkp.pl/

[4] Automapa. Dostępna w World Wide Web: http://www.automapa.com.pl/

prowadzącą do celu (Rysunek 1), jak również poprzez informowanie głosowe o zbliżających się zakrętach i odległościach pozostających do nich. Przydatną cechą systemu jest jego adaptacyjność. W momencie, gdy kierowca przeoczy skręt lub po prostu był on niemożliwy do wykonania z powodów losowych, system automatycznie dokonuje korekty trasy.

Rysunek 1 Przykładowy ekran aplikacji Automapa (nadzór w czasie jazdy)

Rysunek 2 Przykładowy ekran aplikacji Automapa (prezentacja zaplanowanej trasy)

Istnieją także bardziej zaawansowane systemy planowania podróży, które potrafią znaleźć połączenie lotnicze i zaproponować hotel na czas pobytu w docelowym mieście. Jednym z takich systemów jest portal internetowy *Travelocity* [45]. Wadą tego portalu jest obsługiwanie tylko połączeń lotniczych i hoteli tylko dla tych miast, które posiadają lotnisko w swojej okolicy. Daje on jednak możliwość całkiem szerokiego wyboru. Co więcej, czasami jest w stanie pokazać mapkę z zaznaczoną lokalizacją wybranego miejsca noclegu. System ten umożliwia także przeglądanie ofert wyjazdów turystycznych. Niestety zakłada on, iż wylatuje się z Anglii i jako cel można wybrać jakiś obszar atrakcyjny turystycznie, a nie konkretne miejsce (w sumie kilkadziesiąt możliwości). Istnieje więcej stron tego typu. Dla przykładu można podać: *Ebookers*, *Opodo* lub *Traveliada* [34, 40, 44].

2.2 Jak powinien wyglądać idealny system wspomagający podróżowanie?

Jak widać, istnieją już pewne rozwiązania stanowiące pierwsze kroki w kierunku stworzenia *Osobistego Agenta Podróży*. Jednak żadne z nich nie jest idealne, każdemu czegoś brakuje. Jak już wspomniano wcześniej, największym ograniczeniem jest ilość różnorodnych informacji, jakie trzeba by było przetwarzać, aby system mógł spełnić wszystkie stawiane mu oczekiwania. Można się zastanawiać czy jest to jedyny problem, jaki pozostaje do rozwiązania. Oczywiście nie. Osobisty agent powinien być przede wszystkim dostępny dla użytkownika w każdym miejscu, nie tylko, gdy użytkownik znajduje się przy swoim osobistym komputerze[5]. Powinien być w stanie pomóc, gdy użytkownik znajduje się w nieznanym mieście i nie wie jak dojść na umówione spotkanie. Poza tym, dużym udogodnieniem by było, gdyby agent był w stanie monitorować odbywającą się podróż oraz reagować na opóźnienia i inne zdarzenia losowe mające na nią wpływ.

2.2.1 Jak powinno wyglądać planowanie podróży?

Proces planowania podróży nie zawsze może być przeprowadzony z dużym wyprzedzeniem. Zdarza się, że dochodzi do nagłych spotkań, czy też po prostu podczas pobytu na wakacjach decyzje o zwiedzaniu podejmowane są ad hoc. W związku z tym, osobisty agent powinien być tam wraz ze swoim użytkownikiem, cały czas pozostając do jego dyspozycji. Stąd potrzeba, aby mógł działać na urządzeniach przenośnych typu palmtop, czy choćby zwykły telefon komórkowy. Oczywiste jest, że urządzenia te byłyby jedynie interfejsami całego systemu i umożliwiałyby jedynie wprowadzanie zapytań do systemu oraz prezentację proponowanych rozwiązań. Serce systemu znajdowałoby się na ogólnie dostępnych serwerach. Dostęp do nich mógłby być możliwy poprzez już wspomniane urządzenia przenośnie, ale także przy pomocy zwykłego komputera i przeglądarki internetowej. Co więcej wykorzystanie rozwiązania opierającego się

[5] W końcu ma być osobisty!

na architekturze klient-serwer daje użytkownikowi możliwość podłączenia się do systemu, wydania zapytań, których realizacja może zająć pewną ilość czasu, rozłączenia się i odbioru wyników dopiero przy następnej sesji z systemem. Dodatkowo każda sesja może być realizowana przez inne urządzenie o różnych możliwościach prezentacji.

Aby maksymalnie ułatwić korzystanie, łatwo jest sobie wyobrazić, że użytkownik może zadać zapytanie przy pomocy tekstu, wskazać punkty nawigacyjne na mapie, czy też wydać zapytanie głosem [7]. System powinien być też na tyle inteligentny, aby bez trudu rozumieć zapytania podające dokładne współrzędne startu i celu podróży, jak również takie zadawane w języku naturalnym: „*Jak mogę dojechać do najbliższej restauracji*?" lub „*Jak dojść do najbliższego kina, w którym grają film X*?". Widać, że punkty zainteresowania mogą być podawane w różny sposób, niekoniecznie wprost. W przypadku planowania dalszych podróży, użytkownik powinien mieć możliwość określenia innych parametrów takich jak maksymalna ilość przesiadek, czas ich trwania, jakie środki transportu mogą być brane pod uwagę, żądanie tylko przedziałów dla niepalących i jeszcze wiele innych.

Rysunek 3 Przykład wielomodalnej aplikacji wykorzystującej mapy [7]

Każdy z tych aspektów stanowi problem w sam w sobie. O ile samo rozpoznawanie mowy naturalnej nie stanowi już w dzisiejszych czasach problemu, o tyle rozumienie sensu wypowiedzi jeszcze ciągle tak.

2.2.2 Jak powinien wyglądać nadzór nad odbywającą się podróżą?

Inną ciekawą kwestią, jaka rodzi się, gdy mówimy o *Osobistym Agencie Podróży* jest możliwość prowadzenia przez niego nadzoru nad odbywającą się podróżą[6]. Agent powinien mieć zdolność obserwowania postępów, śledzić czy, na przykład, samolot nie jest opóźniony. Gdy dochodzi do takiego opóźnienia, powinien umieć w czynny sposób na nie zareagować. Oznacza to podjęcie przez niego szeregu różnych zadań. Przede wszystkim powinien spróbować tak zmodyfikować plan podróży, abyśmy zdążyli dotrzeć do celu na czas. Ponieważ najprawdopodobniej będzie się to wiązało z wykorzystaniem szybszych środków transportu, a co za tym idzie, zwiększeniem się ogólnych kosztów, decyzję powinien skonsultować z użytkownikiem. Natomiast, gdy nie uda się znaleźć alternatywy mieszczącej się w tych samych ograniczeniach czasowych, powinien podjąć akcje informacyjne skierowane do usługodawców. Chodzi tu, na przykład, o poinformowanie recepcji hotelu, w którym użytkownik zamierza się zatrzymać, o opóźnieniu, aby jego miejsce nie zostało sprzedane komuś innemu.

Oczywiście opóźnienia nie są jedynymi czynnikami wymuszającymi zmiany adaptacyjne w planie przygotowywanym przez agenta. Takie zmiany mogą wymusić również korki w mieście (potrzeba znalezienia objazdu), czy choćby zmiana planów przez samego użytkownika. Dobrze by było, aby nowa wersja planu była maksymalnie podobna do wersji pierwotnej – wykorzystywała maksymalnie już wykupione bilety na połączenia, czy też nawet same rezerwacje (rezygnacja z nich powoduje, iż te miejsca mają mniejszą szansę na wykorzystania przez innych i efekcie zmniejszają ogólną wydajność całego systemu transportu). Wracając jeszcze na chwilę do korków, należy pamiętać o tym, że znajdowanie objazdów w mieście nie jest takim prostym zadaniem, jakby

[6] Pomysł przedstawiony między innymi w [13].

się mogło wydawać. Wbrew pozorom nie wystarczy tylko wyznaczyć trasę, która nie przebiega przez newralgiczne części aglomeracji, ale trzeba również pamiętać, że systemy wspomagające należące do innych użytkowników najprawdopodobniej wyznaczą takie same objazdy (zwłaszcza, gdy wykorzystują to samo oprogramowanie) i wtedy mogą się pojawić nowe korki, a stosowanie objazdu nic nie da...

Monitorowanie może także obejmować okres przed rozpoczęciem podróży. Na przykład, gdy użytkownikowi zależy na maksymalnym ograniczeniu kosztów, można monitorować ceny i jeśli któraś z nich się zmniejszy lub zwiększy, to może się okazać, że to inna z przygotowanych propozycji podróży staje się atrakcyjniejsza finansowo od tej wybranej dotychczasowo. W takim przypadku użytkownik powinien zostać poinformowany o nowej możliwości i mieć możliwość zmiany dotychczasowych planów.

Należy też odpowiedzieć na pytanie: jak monitorowanie miałoby wyglądać od strony technologicznej? Jak agent miałby śledzić podróż swojego użytkownika, skąd wiedzieć gdzie on się aktualnie znajduje, i czy ma on opóźnienie? Nie są to pytania, na które łatwo znaleźć satysfakcjonującą odpowiedź. Ogólnie, w przypadku śledzenia masowych środków transportu, można rozważać dwa podejścia. Pierwszy z nich to monitorowanie fizycznej lokalizacji użytkownika, na przykład, poprzez odczyt wskazań odbiornika GPS i porównywanie aktualnej pozycji z teoretyczną trajektorią [9]. Różnica między położeniem aktualnym a oczekiwanym dawałaby pojęcie aktualnego opóźnienia. To rozwiązanie ma jednak pewne wady. Przede wszystkim, biorąc pod uwagę fakt, iż agent nadzorujący znajdowałby się w innym miejscu, musiałby on być ciągle informowany o wynikach odczytów, a to generowałoby nieustanny ruch spowodowany przekazem informacji. Można temu jednak przeciwdziałać, gdy urządzenie posiadane przez użytkownika jest samo w stanie kontrolować swoje położenie w stosunku do zaplanowanej trajektorii i wysyłać informacji tylko, gdy występują różnice. Innym kłopotliwym problemem jest to, że nie zawsze będzie istniała możliwość transmisji danych (na przykład podczas lotu samolotem). Drugie podejście do monitorowania przebiegu podróży bazuje na innym źródle informacji. Otóż, w niektórych przypadkach, można zdobyć potrzebne informacje,

na przykład, z systemu informacji lotniskowej, która dysponuje informacjami o opóźnieniach poszczególnych lotów. Problemem jest jednak, że systemy, które mogłyby stanowić źródło takich informacji, nie istnieją dla wszystkich środków transportu, albo nie są po prostu dostępne w trybie on-line.

2.3 Personalizacja

Do tej pory dyskutowane były bardziej ogólne zagadnienia dotyczące działania *Osobistego Agenta Podróży*. Nie była jednak poruszana kwestia personalizacji agenta.

Przede wszystkim należy się zastanowić, co to znaczy „personalizacja" w świetle działania agenta podróży? Na pewno oczekiwane jest pod tym pojęciem coś więcej niż tylko posiadanie możliwości nakładania filtrów na szukane rozwiązania - określenie maksymalnej ilości przesiadek, czy też maksymalnego czasu ich trwania, itd. Oczekiwane jest raczej, aby agent – dodatkowo – uczył się upodobań użytkownika, aby poznawał jego preferencje i „zwyczaje" podróżowania. Chodzi tu nie tylko o to, aby potrafił zauważyć, iż jego użytkownik woli jeździć autobusem niż dojeżdżać pociągiem podmiejskim (a co za tym idzie proponować w pierwszym rzędzie plany opierające się właśnie na autobusach, a nie na pociągach), ale również o to, aby w przypadku wyjazdu na dłuższy czas potrafił z gąszczu możliwych hoteli wybrać taki, który będzie mu najbardziej odpowiadał zaoszczędzając użytkownikowi wiele cennego czasu. Jest rzeczą oczywistą, iż takie zachowanie nie dotyczyłoby jedynie hoteli, lecz również restauracji, muzeów, itp.

Warto również zauważyć, iż taka funkcjonalność doradcza byłaby bardzo przydatna, na przykład, w momencie planowania wyjazdu turystycznego. Z reguły, turysta udający się w jakieś miejsce, nie zna wszystkich dostępnych atrakcji i obiektów turystycznych. Bazując tylko na folderach reklamowych będzie w stanie poznać tylko te najbardziej znane i popularne miejsca. Jednak nie znajdzie w nich informacji o tych mniej popularnych, które mogą nie być atrakcyjne dla przeciętnego turysty, ale akurat mogłyby być bardzo ciekawe dla

niego. Agent mógłby układając plan zwiedzania wybrać właśnie takie mniej popularne możliwości.

Pewnym problemem w osiągnięciu takiej funkcjonalności jest - na chwilę obecną – brak odpowiednich repozytoriów wiedzy z potrzebnymi i uporządkowanymi informacjami. Aczkolwiek dostępne są już pewne bazy danych, czy też serwisy on-line, dotyczące - między innymi - hoteli, muzeów i restauracji [33]. Niestety, dane w nich prezentowane nie zawsze zawierają te same rodzaje informacji, a niekiedy mogą nawet podawać sprzeczne dane na ten sam temat.

Personalizacja agenta podróży mogłaby się również przejawiać w jego umiejętności do wyszukiwanie pewnych wzorców w rozkładach codziennych podróży użytkownika, na przykład, do pracy i z powrotem do domu. Ich poznanie umożliwiałoby ich wcześniejsze uwzględnienie w planowanym rozkładzie dnia, dodanie do nich pewnych dodatkowych zadań (na przykład zakupów), czy też przypominanie o braku ważnego biletu.

Oprócz statystycznych obserwacji *Osobisty Agent Podróży* mógłby korzystać także z osobistego organizatora swojego użytkownika. Przypuśćmy, iż użytkownik zleca swojemu agentowi zaplanowanie wyjazdu biznesowego do swojego partnera w interesach. Dla ustalenia uwagi niech będzie to pan Jan Kowalski. Agent zaglądając do osobistego organizatora będzie w stanie ustalić, gdzie znajduje się biuro Kowalskiego oraz pozna aktualny rozkład zajęć na najbliższy czas. Korzystając z tych informacji będzie mógł określić dogodną porę na odbycie podróży - na podstawie odległości będzie mógł oszacować ile potrzeba na nią czasu (godzinę, dwie, a może cały dzień) i poszukać połączeń w odpowiednio długim niezaplanowanym czasie. W ten sposób znajdzie propozycje, które będą mogły być przedstawione użytkownikowi do wyboru (oczywiście po uprzedniej ocenie pod kątem jego upodobań).

2.4 Automatyczne pozyskiwanie informacji

Jak już było to wcześniej zauważone, jednym z istotnych problemów związanych z tworzeniem *Osobistego Agenta Podróży* jest problem gromadzenia olbrzymiej ilości informacji. Wiele informacji dotyczącej podróżowania jest już dostępna w Internecie, jednak są one mocno rozproszone. Różne serwisy zawierają dane dotyczące tylko jakiegoś regionu, albo, na przykład, pojedynczego lotniska. Co gorsza informacje te często są opisywane w różny sposób, zawierają dane inaczej usystematyzowane lub zawierające inne szczegóły. Stąd rodzi się trudne do zrealizowania zadanie integracji tych heterogenicznych danych, co jest konieczne, aby móc tą informację przetwarzać w jednorodnym systemie. Usystematyzowaniem informacji i opisaniem jej w spójny sposób zajmują się ontologie. Tematyka ta zostanie poruszona dalej. Teraz zajmiemy się możliwościami automatycznego pozyskiwanie tych informacji z Internetu nie zgłębiając zbytnio problemu jak je potem zapisywać.

Informacje dostępne w Internecie z reguły są publikowane w postaci stron WWW, gdyż ich odbiorcą jest człowiek. Format HTML i inne nie były projektowane z myślą, że mogą one być przetwarzane przez maszyny. Służyły one raczej za środek umożliwiający dodanie do treści opisu warstwy prezentacji (elementów formatujących ją). Stąd problemy z wydobywaniem informacji prezentowanych przy ich pomocy. Biorąc pod uwagę różnorodność graficznych układów prezentacji danych na stronach (nie mówiąc już o różnej zawartości), stajemy przed smutną koniecznością tworzenia dedykowanych parserów dla każdej strony prezentującej pewne dane. Co gorsza, każda zmiana w układzie graficznym strony najprawdopodobniej będzie oznaczała konieczność modyfikacji już stworzonego parsera. Są jednak prowadzone prace nad automatycznymi agentami potrafiącymi uczyć się zawartości stron (czasem przy pomocy człowieka), które będą sobie radziły z drobnymi modyfikacjami stron WWW, co na pewno zaoszczędzi wiele czasu [30].

Strony WWW, choć najczęściej spotykane, nie są oczywiście jedynymi możliwymi źródłami informacji dostępnymi w Internecie. Internet może również

służyć do propagacji informacji udostępnianej przez bazy danych, na przykład, poprzez serwisy XML-owe. Problem polega na tym, iż w chwili obecnej nie istnieje zbyt wiele zbiorów danych udostępnianych w ten sposób (przynajmniej tych darmowych).

Zakładając, że można dane zbierać w automatyczny sposób, pojawia się kolejny problemem: jak zapewnić spójność danych? Wiadomo, że informacje zdobywane w różnych miejscach mogą się różnić. Gdy się dopełniają nawzajem, jest dobrze; natomiast, gdy są sprzeczne – co najprawdopodobniej będzie sytuacją dominującą – to rodzi się cała seria kolejnych pytań: Jak te dane ze sobą pogodzić? Któremu źródłu ufać bardziej? Co robić, gdy konfliktowe informacje z pewniejszego źródła pojawiają się już po wykorzystaniu informacji gorszych? Jak często odświeżać posiadane informacje? Co robić, gdy źródło nie jest już dostępne?

Zastanawiając się nad automatycznym zbieraniem informacji należy również rozważyć jeszcze jedną kwestię, a mianowicie, czy takie zbieranie informacji nie naruszałoby praw *sui generis* ochrony baz danych innych podmiotów prawnych?

2.5 Ontologie

W poprzednim rozdziale rozważane były kwestie związane ze zbieraniem informacji. Nie poruszono jednak kwestii, jak ją przechowywać (w celu późniejszego przetwarzania). Ponieważ, jak już wiadomo, różne źródła informacji mogą różnie prezentować informacje dotyczące tych samych rzeczy (w szczególności zawierać różne detale), powstawał problem ujednolicenia wiedzy w danym zakresie, na przykład, opisów hoteli, czy też restauracji. Posiadanie uogólnionego sposobu opisu pewnych obiektów (hoteli, restauracji) daje możliwość operowania na tych danych na poziomie ich znaczenia semantycznego. Właśnie takie możliwości dają ontologie.

Ontologia jest pojęciem zapożyczonym z filozofii, gdzie oznacza naukę o bycie, charakterze i strukturze rzeczywistości. W informatyce, ontologie to nic innego jak sformalizowane definicje opisu pewnych pojęć. Definiują one pewnego rodzaju słownik dla danej domeny (na przykład turystyki). Zakładają one, iż każdy obiekt posiada szereg atrybutów, które go definiują. Nie mają one jednak płaskiej struktury, lecz mogą być hierarchiczne i zawierać relacje między poszczególnymi elementami. Należy o nich myśleć raczej jak o grafach. Co więcej, poszczególne elementy (węzły) mogą się odwoływać poprzez relacje do elementów, czy też typów, spoza aktualnego dokumentu (gdy mamy już do czynienia z konkretnymi danymi). Daje to możliwość używania w wielu dokumentach tych samych typów i przez to tworzenia jednolitych systemów danych. Jednak największą korzyścią płynąca z tego jest możliwość odejścia od formatów danych reprezentujących tylko treść (ewentualnie rozszerzoną o informacje warstwy prezentacji) takich jak HTML, czy XML, i przejście do opisu semantycznego znaczenia dokumentów, który będzie już zrozumiały dla maszyn. Będzie, ponieważ dokument nie będzie ze sobą już niósł tylko informacji w postaci zwykłego tekstu, który notabene nie jest zrozumiały dla komputera, ale będzie zawierał pewne dodatkowe informacje pozwalające maszynie ustalić, czym te informacje są, co opisują. Komputer będzie wiedział, że dany fragment tekstu opisuje, na przykład, nazwę hotelu, inny określa dostępne w nim dodatki takie jak basen czy siłowania, a jeszcze inny opisuje cenę doby hotelowej. Dodatkowo będzie świadomy tego, że „cena" doby hotelowej to to samo pojęcie, co, na przykład, „cena" posiłku w restauracji (jest to o tyle istotne, że obliczając koszty, można te dwie wartości do siebie dodać). Mało tego, dzięki możliwości określania relacji między informacjami, będzie mógł powiązać ze sobą fakty, które występują w różnych dokumentach. W ten sposób maszyna będzie mogła operować (w pewnym sensie) już nie ciągiem nic nieznaczących słów, ale pewnym pojęciem; będzie mogła „myśleć" w kategoriach, na przykład, hoteli, a nie tabel w relacyjnej bazie danych, gdzie akurat jedna z nich nosi nazwę „hotele".

Ontologie można opisywać przy pomocy różnych formatów danych. Najpopularniejsze z nich to RDF i OWL [42, 46]. Dają one możliwość łatwego opisu wcześniej wspomnianych zależności między danymi.

Ontologie dają możliwość opisu pewnych pojęć. Można je także łatwo przenieść do świata języków programowania. Ontologicznemu opisowi pewnego pojęcia można przypisać bezpośrednio odpowiadającą mu klasę w dowolnym z języków obiektowych (zawierającą te same informacje). Od tego momentu operowanie na nim jest już nieporównywalnie łatwiejsze i pozwala skupić się programiście na istotniejszych zagadnieniach, niż operowanie pojęciami w świecie tabel relacyjnych baz danych.

2.6 Badania pod przewodnictwem Craig'a Knoblock'a

Naukowcy próbują wcielać w życie różne pomysły związane z szeroko rozumianym planowaniem podróży. Jedną z takich grup badawczych jest grupa pracująca pod przewodnictwem Craig'a Knoblock'a[7]. Podczas swoich prac grupa ta opublikowała szereg dokumentów i wyników działania programów związanych z planowaniem podróży oraz automatycznym zbieraniem informacji dostępnych w Internecie. W dokumentach publikowanych przez nich na stronach internetowych można przeczytać między innymi o dwóch programach, które są szczególnie interesujące z punktu widzenia planowania podróży. Są to *Heracles* i *Theseus* [38, 43].

„*Heracles to system ramowy do tworzenia asystentów informacyjnych, którzy wspomagają budowę aplikacji dedykowanych dla określonej domeny, które to wydobywają i integrują dane, aby wspomóc określone zadanie. Nie są to jednak silniki wyszukujące, ale raczej aplikacje organizujące i integrujące dane w celu wspomożenia określonego zadania.*"[8] Jego głównym autorem jest Craig Knoblock. Jednym z jego zastosowań (opisanym jako przykład jego użycia) jest system wspomagający podróżowanie. Jego założeniem jest uniknięcie konieczności przeglądania wielu stron WWW w poszukiwaniu różnych informacji potrzebnych do zaplanowania podróży (takich jak hotele, połączenia lotnicze, czy

[7] Craig Knoblock, Ph. D. (Information Sciences Institute, University of Southern California)

[8] Opis projektu Heracles. Tłumaczenie własne.

prognoza pogody) poprzez zebranie najbardziej wartościowych z dostępnych źródeł i połączenie ich w jeden zintegrowany system. Zadaniem tego systemu jest pomoc w planowaniu wyjazdów w celach biznesowych.

W jednym z dokumentów został opisany system o nazwie *Travel Assistant*, który sam pobiera listę zaplanowanych spotkań z osobistego kalendarza i po wybraniu jednego z nich przez użytkownika pobiera resztę informacji: datę oraz lokalizację spotkania, a także prognozę pogody na czas jego trwania [13]. Dokonuje również rekomendacji dotyczących sposobu dotarcia na miejsce spotkania bazując na odległości, którą trzeba pokonać. Może to być dojazd prywatnym samochodem, taksówką lub lot samolotem. Podpowiada też, w przypadku lotu, czy bardziej się opłaca jechać samochodem na lotnisko i zapłacić za postój na parkingu, czy też wziąć taksówkę. Oprócz tego, wybiera najbliższy hotel w okolicy spotkania i może zoptymalizować propozycje podróży pod kątem czasu lub cen. Jako ciekawą opcję, wspomnianą już wcześniej, udostępnia agentów odpowiedzialnych za nadzorowanie podróży, którzy między innymi, mogą poinformować hotel o spóźnieniu użytkownika. Prezentowany system oznacza się także ciekawym interfejsem umożliwiającym w paru krokach zaplanować całą podróż. Interfejs ten reaguje dynamicznie na wybory wykonywane przez użytkownika i, na przykład, automatycznie przelicza całkowite koszty podróży, gdy użytkownik zmieni choć jeden jej detal (dokonuje także nowych rekomendacji, gdyż może się okazać, że po zamianie jednego lotu na drugi może już bardziej opłacać się jazda taksówką a nie własnym pojazdem).

Pewnym ograniczeniem prezentowanego systemu jest to, iż umożliwia on stworzenie planu, który tylko obejmuje prosty dojazd. Zakłada on, iż podróż składa się albo tylko z samej jazdy samochodem, albo z lotu samolotem połączonego z dojazdami samochodami do/z lotnisk. Takie ograniczenie jest jednak zrozumiałe, gdy weźmiemy pod uwagę fakt, iż jest to tylko system mający zademonstrować pewne możliwości, a nie służyć jako rozwiązanie produkcyjne.

Jak już wspomniano, system ten wykazuje dynamiczne reakcje na decyzje podejmowane przez użytkownika. Mogą to być zarówno zmiany dotyczące rekomendacji, jak i efekty zmiany wybranego połączenia (na przykład

konieczność wyświetlenia innej mapy). Aby było to możliwe system korzysta z systemu rozumowania z ograniczeniami. Dzięki niemu może zawęzić wybór możliwych opcji do tych, które są adekwatne do innych (nadrzędnych) lub uruchomić odpowiednich agentów odpowiedzialnych za zdobycie potrzebnych informacji z Internetu. System rozumowania jest przedstawiony jako sieć ograniczeń (zbiór zmiennych wraz ze skierowanymi ograniczeniami - predykatami, które zachodzą między nimi, na przykład: obliczanie długości pobytu, czy całkowitej ceny za parking). Ewaluacja predykatów jest wykonywana w momencie, gdy wszystkie powiązanie z nim zmienne mają już przypisaną wartość. Zmiana wartości niektórych zmiennych może pociągnąć za sobą konieczność ponownej ewaluacji pewnych predykatów – jest to propagacja ograniczeń, która umożliwia dynamiczne reakcje na działania podejmowane przez użytkownika lub informacje napływające od agentów monitorujących. Dokładniejszy opis działania systemu jest także dostępny w dokumencie „*Mixed-initiative, multi-source information assistants*" [21].

Drugim z interesujących systemów z punktu widzenia planowania jest *Theseus*. System ten odpowiada za wydobywanie informacji ze źródeł znajdujących się w Internecie. Jego głównym autorem jest Greg Barish. Zawiera on w sobie podsystem planujący i wykonujący zapytania do zdalnych heterogenicznych źródeł danych. Oprócz samego wydobywania danych, zakłada on również możliwość monitorowania stanu zawartości tych źródeł i podejmowaniu pewnych akcji, gdy się on zmieni.

Sercem systemu jest moduł wykonawczy realizujący zadania równolegle i asynchronicznie, a jego działanie opiera się na grafie przepływu danych. Posiada on także własny język służący do opisu planów wykonania (między innymi umożliwia warunkowe i powtarzające się czynności). Graf opisujący plan składa się węzłów będących operatorami oraz krawędzi określających przepływ sterowania i/lub danych. Każdy operator może wygenerować wiele danych, niekoniecznie tylko jedną wartość, które są przekazywane dalej i pobudzają kolejne operatory. Operatory są wykonywane równolegle wszędzie tam, gdzie jest to możliwe. Zapewnia to zwiększenie wydajności agenta, który będzie realizował

dany plan. Wykonywanie planu kończy się, gdy nie ma już aktywnych operatorów – czyli w momencie, gdy wszystko zostało przetworzone do końca.

Aby zwiększyć wydajność *Theseusa*, stosowane są przewidywania co do kolejnych zapytań i predykatów, co pozwala na szybsze uzyskanie wyników poprzez wcześniejsze i równoległe wykonanie kwerend [2, 3, 4, 5]. Ogólnie polega to na przewidywaniu kolejnych zapytań i ich wykonywaniu z wyprzedzeniem lub – odpowiednio – na przewidywaniu wartości predykatów. Jeżeli przewidywania były słuszne, to zaoszczędza się na czasie, jeśli nie, to i tak nie traci się nic oprócz kosztów wykonania kwerend lub predykatów z wyprzedzeniem.

Do samego wydobywania informacji ze stron WWW wykorzystywane są dedykowane opakowywacze (ang. wrappers), które znają ich układ, rozmieszczenie w nich informacji oraz ich logiczne powiązania między sobą i dzięki temu są stanie wydobyć z nich odpowiednie dane, które są następnie konwertowane do wewnętrznego formatu używanego przez aplikację.

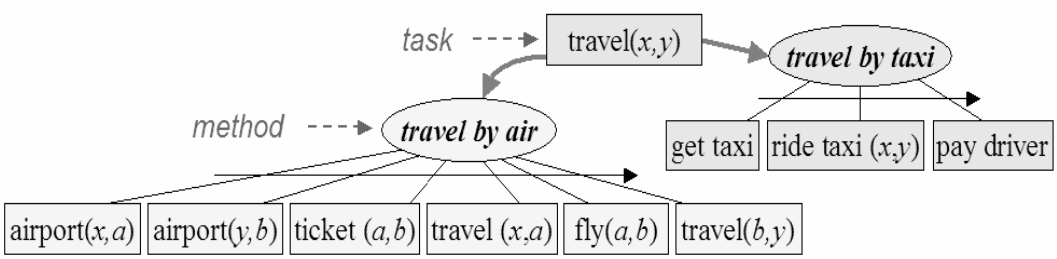
2.7 Planowanie przy pomocy Hierarchicznych Sieci Zadań

Opisana w *Heraclesie* metoda planowania podróży ma pewną istotną wadę. Mianowicie zakłada ona, że podróż można odbyć pokonując praktycznie cały dystans przy pomocy samolotu, a całą resztę przy pomocy dojazdów na i z lotnisk prywatnym samochodem lub taksówką. Niestety często nie jest to możliwe. Cel podróży, nie musi mieć lotniska w swojej okolicy i może się okazać, że trzeba użyć większej ilości środków transportu, na przykład, pokonać odcinek trasy pociągiem albo autobusem. Użytkownikowi może też zależeć na znalezieniu trasy dojazdu w mieście przy pomocy transportu miejskiego – autobusów i tramwajów. Te i inne przypadki nie dają się tak łatwo rozwiązać ze względu na możliwą większą ilość przesiadek i ilość koniecznych do wykorzystania środków transportu. Potrzebny jest sposób, który umożliwiłby *Osobistemu Agentowi Podróży* znalezienie takiego bardziej złożonego planu. Nie jest to oczywiście zadanie łatwe. Do rozpatrzenia jest bardzo wiele możliwości – trzeba ustalić

czym, skąd i dokąd się przemieszczać w taki sposób, aby wszystkie etapy podróży układały się razem w sensowną i spójną całość realizującą zamierzony cel. Właśnie ze względu na bardzo dużą ilość możliwości metody przeszukiwania przestrzeni poszukiwań stosowane w klasycznej sztucznej inteligencji nie znajdują tu zastosowania praktycznego. Okazuje się, że z takimi problemami planistycznymi można sobie poradzić wykorzystując *Hierarchiczne Sieci Zadań* (ang. *Hierarchical Task Networks*).

Ogólne zasady planowania z wykorzystaniem tych sieci można znaleźć, między innymi, w pracach Dany Nau [22, 27]. Głównym założeniem tych metod jest dekompozycja zadania na coraz mniejsze podzadania przy pomocy zdefiniowanych metod dzielenia, aż do momentu uzyskania podziału zadania na podzadania elementarne, które wiadomo jak już wykonać. Pomysł działania w ten sposób nie jest nowy, bo liczy już sobie prawie trzydzieści lat[9]! Opiera się on na obserwacji działania ludzi, którzy rozwiązując jakiś bardziej skomplikowany problem, wcale nie rozpatrują wszystkich możliwości, tylko rozbijają go na mniejsze podproblemy, z którymi potrafią już sobie poradzić bazując na swoich doświadczeniach z przeszłości. Dla demonstracji posłużmy się przykładem (Rysunek 4). Jak widać, zadanie podróżowania może zostać rozbite przy pomocy metod *travel by air* (podróż samolotem) lub *travel by taxi* (podróż taksówką) na szereg podzadań, które wykonane w odpowiedniej kolejności zapewnią rozwiązanie początkowego problemu. Oczywiście metod podziału może być więcej. W tym przypadku mogłyby to być metody oparte, na przykład, o pociąg, statek, czy też chodzenie. Widać także, że może pojawić się również pewna rekurencja w podziale. Na przedstawionym przykładzie dzieje się tak, gdy użyto metody *travel by air*. W tym przypadku pojawia się konieczność rozwiązania dwóch podproblemów dojazdów do lotnisk.

[9] Sacerdoti oraz Tate (1977 rok)



Rysunek 4 Schemat dekompozycji zadań [23]

Oczywiste jest, że stosowanie metod podziału nie zawsze jest możliwe. Aby móc skorzystać z jakiejś metody podziału, muszą być spełnione jej warunki początkowe. Na przykład, taksówka nie nadaje się do pokonywania znacznych odległości, a samolot do pokonywania tych mniejszych, natomiast do zakupu biletu potrzebna jest odpowiednia ilość pieniędzy, itd. Co więcej, trzeba brać pod uwagę pewne interakcje zachodzące między poszczególnymi zadaniami. Podczas planowania podróży może to być, na przykład, konieczność wyjechania z domu odpowiednio wcześnie, aby zdążyć na pociąg, który odchodzi o określonej godzinie. Ze względu na wspomniane warunki początkowe i zależności może się okazać, że w pewnym momencie nie można skorzystać już z żadnej metody podziału, a ciągle pozostaje do rozwiązania pewne zadanie złożone. W takim przypadku konieczne jest wycofanie się z wykonanego podziału i na odpowiednio wyższym poziomie w drzewie wyborów skorzystanie z innej metody podziału (tzw. *backtracking*).

Wykorzystanie planowania opartego na *Hierarchicznych Sieciach Zadań* ma też tą zaletę, że wykonywane dekompozycje są w porządku absolutnym, tzn. uzyskane zadania proste są już w odpowiedniej kolejności (czyli w takiej, w jakiej powinny być wykonane). Część z metod sztucznej inteligencji wykorzystywanych w planowaniu opierająca się na rozwiązywaniu problemów częściowych generuje - w efekcie takiego podejścia - rozwiązania z porządkami częściowymi, co rodzi problem późniejszego odpowiedniego wymieszania rozwiązań cząstkowych między sobą w taki sposób, aby tworzyły one razem możliwą do wykonania całość. Podobnie jest w przypadku metod opartych na *backward chainingu*, kiedy to, z reguły, nie jest znany aktualny stan świata i przez to istnieją problemy z przeprowadzaniem rozumowań logicznych na operatorach planowania (gdyż nie

można łatwo określić czy można ich użyć). Przytoczone problemy można rozwiązać, jednak jest to z reguły kosztowne. Wracając jednak do *Hierarchicznych Sieci Zadań* to, że dekompozycje są w porządku absolutnym, umożliwia w każdym kroku algorytmu znanie aktualnego pełnego stanu rozpatrywanego świata. Dzieje się tak, ponieważ można określić wpływ już wykonanych akcji na świat poprzez obserwację jednoznacznych zmian od stanu początkowego od stanu aktualnego. Dzięki temu zawsze jest się w stanie skorzystać z pełnej wiedzy o świecie, a co za tym idzie, ze wszystkich metod podziału oraz ze wszystkich reguł, które umożliwiają obcinanie gałęzi przeszukiwanej przestrzeni możliwości. W efekcie, metody planowania oparte o *Hierarchiczne Sieci Zadań* dla części zagadnień często dają znacząco lepsze rezultaty w stosunku do innych szeroko stosowanych metod.

3 Travel Support Project

W poprzednich rozdziałach niniejszej pracy zostały przedstawione różne kwestie związane z zagadnieniem *Osobistego Agenta Podróży* z naciskiem położonym na te związane z planowaniem. Każda z nich jest jednak tylko pewnym elementem większej całości. Różne przytoczone prace koncentrują się na różnych kwestiach działania takiego agenta. Żadna jednak, co jest zrozumiałe ze względu na rozmiar całego problemu, nie obejmowała wszystkiego. W pewnym momencie trzeba jednak pójść dalej i spróbować zaprojektować rozwiązanie, które będzie starało się łączyć w sobie jak najwięcej z tych aspektów. W dalszej perspektywie, należy spróbować zbudować odpowiedni system realizujący w pierwszej wersji przynajmniej te najważniejsze ze stawianych celów. W późniejszym czasie można go sukcesywnie rozwijać i rozszerzać jego możliwości.

Cel ten jest realizowany w ramach projektu *Travel Support Project* nadzorowanego przez Marcina Paprzyckiego [14]. Przygotowywane rozwiązanie ma charakter wieloagentowego systemu rozproszonego. System ma być zdolny zbierać w ontologicznie zorientowany sposób dane z dziedziny turystyki dostępne w Internecie i wykorzystywać je do wspomagania procesu planowania podróży. Będzie można go wykorzystać do wyszukiwania obiektów, takich jak hotele czy restauracje lub wykorzystać do automatycznego przygotowania kompletnego planu podróży. Będzie także posiadał szerokie zdolności personalizacyjne.

Jak już zostało to wspomniane, system będzie posiadał charakter wieloagentowy. Różne rodzaje agentów będą odpowiedzialne za realizację wszystkich zadań w systemie. Każdy użytkownik będzie posiadał swojego osobistego agenta (ang. *Personal Agent*), który będzie go obsługiwał poprzez delegowanie odpowiednich zadań do innych specjalistycznych agentów (na przykład, wyszukujących odpowiedni hotel lub planujących podróż z punku A do punktu B). Agenci będą się także zajmowali utrzymywaniem repozytoriów wiedzy (zdobywaniem nowych informacji oraz dbaniem o aktualność tych już posiadanych) oraz personalizacją interakcji z systemem.

W ramach niniejszej pracy zostało przygotowane planowanie podróży. Proces planowania będzie bazować na danych dostarczanych przez inne moduły systemu tworzonego w ramach *Travel Support Project.*.

4 Część praktyczna

4.1 Wprowadzenie

W części praktycznej pracy skoncentrowano się na przygotowaniu algorytmu umożliwiającego zaplanowanie podróży między dwoma dowolnymi punktami zakładając możliwość wykorzystania wielu środków transportu podczas drogi. Znajdowanie sposobu przemieszczania się z punku A do B jest kluczowym zagadnieniem niezależnie od odległości między tymi punktami. Realizacja tego celu umożliwi przygotowanie planu zarówno dla turysty udającego się na wypoczynek na plaży na drugim końcu świata, jak i dla biznesmena, który wyjechał w interesach do nieznanego sobie miasta i potrzebuje znaleźć drogę ze swojego hotelu do biura swoich partnerów handlowych. Wspomniane punkty A i B mogą być podane wprost przez samego użytkownika lub, na przykład, być wynikiem jego zapytania o najbliższe kino. Dodatkowymi celami są wymogi, aby algorytm był na tyle elastyczny, aby mógł się dać relatywnie prosto rozszerzać o nowe środki transportu oraz aby był łatwy do równoległego wykonywania, gdyż w przyszłości może się okazać, że obliczenia równoległe będą konieczne dla szybkiego uzyskania wyników. Pożądane jest również, aby był w stanie brać pod uwagę pewną wiedzę ekspercką, na przykład, informacje, że z Anglii są bardzo tanie połączenia lotnicze do Skandynawii.

Prace nad całym projektem są prowadzone przez wiele osób, a ponieważ ich prace toczą się równolegle do pisania niniejszej pracy, to nie istniała możliwość skorzystania z ich modułów oprogramowania. Stąd pojawiła się konieczność wykorzystania własnych rozwiązań symulujących te brakujące. Przygotowane rozwiązanie zostało zaplanowane tak, aby przejście na współpracę z modułami nieosiągalnymi na chwilę obecną było bezproblemowe. Zostało to osiągnięte przez odpowiednią modułowość architektury i zdefiniowanie odpowiednich interfejsów. Więcej szczegółów na ten temat znajdzie się dalej, w rozdziale poświęconemu implementacji (rozdział 4.4.1).

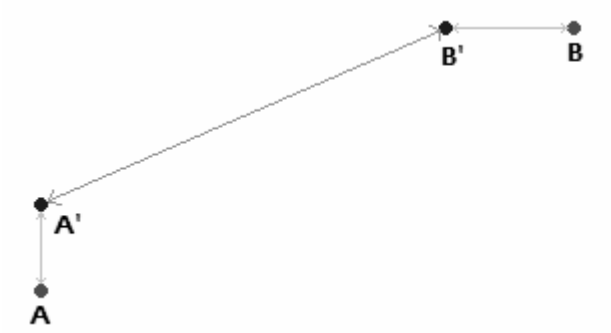
4.2 Przedstawienie rozwiązania problemu planowania podróży

Jak można się spodziewać, planowanie podróży nie jest zadaniem prostym. Aby stworzyć plan, należy wziąć pod uwagę wiele czynników, takich jak ograniczenia stawiane przez użytkownika, czy mnogość źródeł informacji, z których trzeba wydobyć tylko te istotne i ułożyć z nich wartościowy plan realizujący postawione zadanie przemieszczenia się użytkownika z jednego punktu do drugiego. Metodą rozwiązania tego problemu może być obserwacja postępowania ludzi planujących podróż. Ustalają oni najpierw dość ogólnie skąd, dokąd i czym będą podróżować, a dopiero potem zaczynają szukać już konkretnych połączeń, które realizują wcześniej zdefiniowaną ogólną marszrutę. Właśnie taka koncepcja może posłużyć do rozwiązania problemu planowania podróży. Najpierw należy ustalić podział podróży na etapy, które zdefiniują wykorzystywane środki transportu oraz lokacje, między którymi będzie się nimi podróżować, a następnie poszukać konkretnych połączeń implementujących te etapy (przy spełnieniu dodatkowych ograniczeń narzuconych przez użytkownika). Oba kroki algorytmu zostaną teraz przedstawione dokładniej.

4.2.1 Etap I – podział podróży na etapy

Aby ustalić podział podróży na etapy, można wykorzystać rozwiązanie opierające się na opisanych już wcześniej *Hierarchicznych Sieciach Zadań* i pomyśle istnienia głównego etapu podróży zapożyczonym z przykładu zastosowania, także opisanego już wcześniej, *Heracles'a*. Ogólnie rzecz biorąc można założyć, że problem podziału podróży z punktu A do B na etapy można rozwiązać rekurencyjnie w następujący sposób: najpierw ustalić główny etap (z A' do B'), a następnie rozwiązać dwa podzadania podziału podróży z punktu startowego do początku głównego etapu (z A do A') oraz z jego końca do punktu końca podróży (z B' do B). Ilustrację tego opisu można znaleźć na rysunku poniżej (Rysunek 5). Rozwiązywanie rekurencyjne można zakończyć po osiągnięciu maksymalnego poziomu głębokości wywołań rekurencyjnych. Można go wyznaczyć na podstawie maksymalnej ilości przesiadek – każdy wybór etapu głównego oznacza co najmniej dwie przesiadki (w punktach A' i B'). Drugim

powodem przerwania rekurencji jest brak dalszych możliwości podziału, który będzie skutkiem ograniczeń, jakie wynikają z zależności między kolejnymi etapami oraz zależności między lokalizacją początku i końca, a środkiem transportu. Od razu widać, że głównym problemem przy takim podejściu będzie wybór wspomnianych głównych etapów. Trzeba ustalić, jakimi środkami transportu będą one pokonane oraz gdzie te etapy będą się zaczynać i gdzie kończyć (problem wyznaczenia punktów A' i B').



Rysunek 5 Podział odcinka na etapy

Problem ten można rozwiązać podejmując próbę wyznaczenia głównego etapu przy pomocy tych środków transportu, o których nie możemy z całą pewnością powiedzieć, że dla danych dwóch punktów (początku i końca podróży) na pewno nie da się przemieścić między tymi punktami przy ich pomocy, czyli tymi, dla których nie ma żadnych przeciwwskazań, aby z nich skorzystać. Dlatego też potrzebne są pewne reguły mówiące czy dla dwóch danych punktów dany środek transportu jest odpowiedni. Reguły takie można stworzyć. Ich dokładniejszy opis znajdzie się w rozdziale 4.2.1.2. Na razie wystarczy zauważyć, że ich wykorzystanie jest jednym ze środków ograniczenia niepotrzebnych przeszukiwań w II etapie algorytmu planowania podróży (implementacji wyznaczonych etapów), gdzie występuje wiele kosztownych zapytań do bazy danych.

Mając do dyspozycji listę potencjalnych środków transportu - stworzoną przy pomocy reguł - można dla każdego z nich ustalić początek i koniec potencjalnego głównego etapu. Aby tego dokonać wystarczy wybrać *n* najbliższych stacji dla danego środka transportu w pobliżu zarówno początku, jak

i końca podróży (rozdział 4.2.1.3). Dla środków transportu, dla których nie istnieje pojęcie stacji, można przyjąć, że istnieje jedna wirtualna stacja zlokalizowana w tym samym miejscu co początek (lub koniec) etapu. Dalej można przyjąć, że główny etap będzie przebiegał między każdą możliwą parą stworzoną przez jedną stację leżącą niedaleko początku podróży i jedną leżącą niedaleko jej końca (dla środków transportu bez stacji etap główny pokryje cały rozpatrywany odcinek). Branie pod uwagę wszystkich takich kombinacji stwarza wiele możliwości do późniejszego rozpatrzenia, co daje odpowiednio duży potencjał różnorodności, aby właśnie potem znaleźć pewne wartościowe połączenia (nie wszystkie możliwości zaowocują w etapie II znalezieniem odpowiednich połączeń implementujących odpowiednie etapy). Co więcej, działanie na zasadzie par stacji daje możliwość dodania na tym etapie planowania pewnych dodatkowych par stacji, które już nie muszą być rozpatrywane na zasadzie każda z każdą, ale tylko same między sobą. Takie dodatkowe pary mogą być wykorzystane do wsparcia algorytmu planowania wiedzą ekspercką. Na przykład, pracownicy biur podróży wiedzą, że czasami może się opłacać mocno nadłożyć drogi, ale dzięki promocjom znacznie zaoszczędzić na kosztach. System komputerowy zazwyczaj w swoich poszukiwaniach nie znalazłby takiej możliwości, gdyż musi maksymalnie ograniczać ilość przeglądanych opcji, aby działać w akceptowalnym czasie i w związku z tym nie rozpatruje zbyt długich tras wiodących „na około”. Dodanie par stacji w tym momencie może rozwiązać taki problem, gdyż algorytm zostanie zmuszony to ich rozpatrzenia (rozdział 4.2.1.4).

W ostatnim etapie działania, gdy jest już gotowa lista wszystkich możliwych podziałów, można ją jeszcze poddać procesowi ulepszania usuwając z niej rozwiązania jakościowe gorsze od innych (rozdział 4.2.1.5). Kryteriami oceny mogą być, na przykład, długość wyznaczonej trasy (z pominięciem podziałów zawierających wiedzę ekspercką), czy też długość odcinków niezaplanowanych (te mogą powstać, gdy w pobliżu początku lub końca podróży nie ma stacji żadnego środka transportu).

4.2.1.1 Opis działania etapu I algorytmu planowania

Rozdział ten zawiera dokładny opis I etapu algorytmu planowania. Ponieważ algorytm dzielący działa na zasadzie rekurencji zostanie opisane pojedyncze wywołanie jego funkcji rekursji. Jej pseudokod można znaleźć w Załączniku I. Pewne szczegóły tej funkcji zostały dokładniej opisane w kolejnych rozdziałach.

Wywołania funkcji rekursji rozpoczyna się od zainicjowania nowej *listy podziałów*. Następnie jest sprawdzane, czy wywołanie nie jest na zbyt głębokim poziomie. Jeśli tak, to zwracana jest pusta lista. W przeciwnym przypadku, przy pomocy reguł doboru środków transportu, konstruowana jest ich potencjalna lista (rozdział 4.2.1.2). Kolejne operacje są wykonywane dla każdego z wybranych środków transportu.

Wybierane jest *n* najbliższych stacji dla danego środka transportu (rozdział 4.2.1.3) w pobliżu zarówno początku jak i końca dzielonego odcinka (otrzymujemy dwie listy: *stacje początkowe* i *stacje końcowe*). Z nich odrzucane są te stacje, które są stacjami leżącymi dokładnie na początku lub końcu dzielonego odcinka (otrzymujemy listy: *wykorzystywanych stacji początkowych* i *wykorzystywanych stacji końcowych*). Wszystkie one są oznaczane do późniejszego parowania ze sobą nawzajem. Dla *wykorzystywanych stacji początkowych* wyznaczane są rekurencyjnie podziały odcinków między początkiem dzielonego odcinka a tymi stacjami, a dla *wykorzystywanych stacji końcowych* podziały odcinków między tymi stacjami a końcem dzielonego odcinka (otrzymywane są odpowiednio: *podziały początkowe* i *podziały końcowe*). Następnie pobierana jest lista par stacji od modułu z wiedzą ekspercką (rozdział 4.2.1.4). Są one dodawane do odpowiednich list *stacji wykorzystywanych* na tej samej zasadzie, co zwykłe stacje (pod warunkiem, że listy zwykłych stacji nie mogą już wygenerować takiej pary). Podobnie, sprawdzane jest czy nie są to stacje leżące dokładnie na początku lub końcu dzielonego odcinka oraz są dla nich generowane *podziały początkowe* i *końcowe*. Stacje te są oznaczane do późniejszego parowania tylko ze sobą nawzajem. Dalsze operacje są wykonywane dla każdej stacji z *listy wykorzystywanych stacji*

początkowych i dla każdej stacji z listy *wykorzystywanych stacji końcowych* (dla każdej iteracji oznaczamy aktualne stacje jako *stacja początkowa* i *stacja końcowa*).

Sprawdzane jest czy *stacja początkowa* i *stacja końcowa* mogą zostać sparowane na podstawie ich znaczników parowania. Jeśli nie mogą, to następuje przejście do kolejnych *stacji początkowej* i *stacji końcowej*. W przeciwnym wypadku sprawdzane jest czy połączenie między tymi dwoma stacjami nie jest wbrew regułom doboru potencjalnych środków transportu. Jeśli reguły nie odrzucają takiej pary, to tworzona jest kopia *podziałów początkowych* dla *stacji początkowej* i do tych kopii dodawany jest na końcu kolejny etap (aktualny etap główny – pomiędzy *stacją początkową* i *stacją końcową*, z użyciem aktualnego środka transportu). Wynikowe podziały są umieszczane na liście *podziałów z etapem głównym*. Podziały te są łączone z *podziałami końcowym*i dla *stacji końcowej* – dla każdego *podziału końcowego* jest tworzona kopia wszystkich *podziałów z etapem głównym* i jest on dodawany do każdej z nich na końcu. Wynikowe podziały są dodawane do *listy podziałów* pod warunkiem, że nie łamią one ograniczenia na maksymalną ilość przesiadek.

W tym momencie są już wyznaczone wszystkie podziały danego odcinka. Wynikowa *lista podziałów* jest jeszcze poddawana obróbce przed zwróceniem jej jako wynik działania funkcji rekursji (rozdział 4.2.1.5):

- Usuwane są podziały, które nie zaczynają się blisko początku dzielonego odcinka i które jednocześnie nie kończą się blisko końca dzielonego odcinka chyba, że oznaczałoby to usunięcie wszystkich wygenerowanych podziałów. W przypadku, gdy nie istnieją takie podziały, warunek testu jest osłabiany – wystarczy, aby bliskość była zachowana tylko z jednej strony podziału. Gdy i takich podziałów nie ma, zostawiane są wszystkie podziały z listy.
- Minimalizowane są długości niezaplanowanych odcinków (jednocześnie z obu stron podziału lub tylko jednostronnie, w zależności od sytuacji z poprzedniego punktu).

- Usuwane są podziały znacząco dłuższe niż pozostałe (usunięcie ekstremalnych przypadków).
- Usuwane są podziały zawierające środki transportu, które nie zostały dopuszczone przez użytkownika chyba, że oznaczałoby to usuniecie wszystkich podziałów.

Po powrocie z wywołania pierwszopoziomowego ze zwróconej listy podziałów usuwane są powtarzające się podziały, które mogły się pojawić ze względu na różną kolejność wyboru głównych etapów na różnych poziomach wywołania. Kilka różnych ścieżek wyboru może doprowadzić do wygenerowania takiego samego podziału.

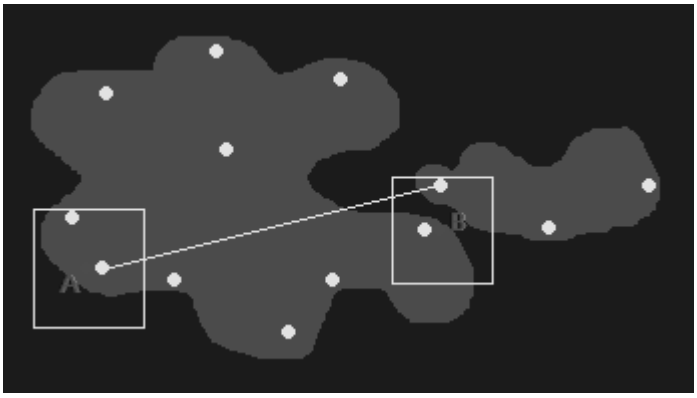
4.2.1.2 Reguły wyboru potencjalnych środków transportu dla dwóch danych punktów

Jak zostało to wcześniej zauważone, wybór możliwych środków transportu (a właściwie definitywne wykluczenie pewnych z nich) dla głównego etapu umożliwia uniknięcie w drugim etapie planowania przeglądania możliwości, które na pewno nie dadzą żadnych rezultatów, ponieważ, na przykład, z całą pewnością nie ma połączenia lotniczego między dwoma pobliskimi miastami leżącymi na tym samym lądzie, albo połączenia kolejowego między dwoma wyspami. Umożliwia to również uniknięcie konieczności generowania podziałów rekurencyjnych w I etapie dla odrzuconych możliwości. Aby stworzyć taką listę środków transportu, którymi możliwe jest przemieszczanie między dwoma danymi punktami, należy użyć reguł. Mogą one zarówno stwierdzać, że w danym przypadku jeden środek transportu może zostać użyty, jak również, że inny nie. Jako takich reguł można użyć, między innymi, prostych zdań bazujących na odległości między dwoma punktami i porównywaniu jej z minimalną i maksymalną odległością specyficzną dla konkretnego środka transportu. Ma to sens o tyle, że nikt nie będzie chciał pokonywać na piechotę 50 km odcinka, albo lecieć samolotem do pobliskiego miasta. Każdy środek transportu nadaje się bardziej niż inne do pokonywania pewnych dystansów. Oczywiście często będzie więcej niż jedna możliwość - na przykład, jazda autobusem lub taksówką - ale już

spacer i lot samolotem, raczej nigdy nie wystąpią jednocześnie. Aby korzystać z takich reguł, wystarczy tylko znać współrzędne obu punktów. Jednak, aby stworzyć bardziej wyrafinowane reguły, trzeba już dysponować większą ilością informacji - danych o położeniu geograficznym takich jak kontynent czy wyspa, na których dany punkt się znajduje. Takie dwie dodatkowe informacje są wystarczające, aby zbudować bardziej zaawansowane reguły, które mówią, na przykład, że nie można jechać pociągiem z miasta w Europie do miasta w Ameryce Północnej albo, że nie da się przejechać samochodem z jednej wyspy na Hawajach na drugą. Reguły mogą także być pomocne w sytuacjach wyjątkowych – dla przykładu, można podać tunel pod Kanałem La Manche – normalnie połączenie między Anglią i Francją zostałoby odrzucone, gdyż są one oddzielone morzem, ale dzięki tunelowi specjalna reguła może dopuścić takie połączenie.

Jak widać, w przedstawionym dalej pseudokodzie wcześniejszego opisu (Załącznik I), pierwsze wywołanie algorytmu nie przewiduje posiadania informacji o stacjach, które to leżą na współrzędnych danych jako dwa pierwsze parametry. Jest tak, gdyż w zastosowaniu praktycznym na początku będą z reguły znane tylko dane pochodzące, na przykład, z GPS o aktualnej pozycji użytkownika lub o lokacji jego mieszkania czy też biura. Nawet, jeśli użytkownik będzie się znajdował na stacji, to i tak zostanie ona wykryta jako jedna z tych najbliższych. To, że pierwsze dane o położeniu będą pochodziły ze źródła typu GPS, będzie oznaczało, że z reguły nie będą znane dodatkowe informacje o kontynencie oraz wyspie, na której znajduje się użytkownik. W związku z tym nie będzie można skorzystać z dodatkowych reguł. W pseudokodzie znajduje się także, w jego dalszej części, ponowne sprawdzenie czy pokonanie odcinka będącego kandydatem na główny etap pewnym środkiem transportu nie jest sprzeczne z tymi regułami. Jest to robione ponownie z uwagi na fakt, iż w tamtym miejscu mamy już do czynienia ze stacjami, które już najprawdopodobniej będą posiadały dodatkowe informacje o położeniu geograficznym i możliwe, że pozwolą na zastosowanie reguł. Mogło się także zdarzyć, że wybór n stacji leżących najbliżej początku i końca dzielonego odcinka wybrał, na przykład, stacje leżące na różnych wyspach. Są to sytuacje ekstremalne, co nie zmienia jednak faktu, że możliwe – ponowne sprawdzenie zgodności z regułami wykluczy tworzenie etapów między takimi stacjami. Przypadek taki podróży z punktu A do

B został zaprezentowany poniżej (prostokąty oznaczają wybrane stacje jako te najbliższe punktom A i B):



Rysunek 6 Przypadek, gdy ponowne zastosowanie reguł doboru środków transportu odrzuca wygenerowaną parę stacji

Zaznaczone połączenie (linia między stacjami) oznacza połączenie, które jest wbrew regułom, gdy środkiem transportu jest, na przykład, pociąg lub inny środek transportu podróżujący po lądzie.

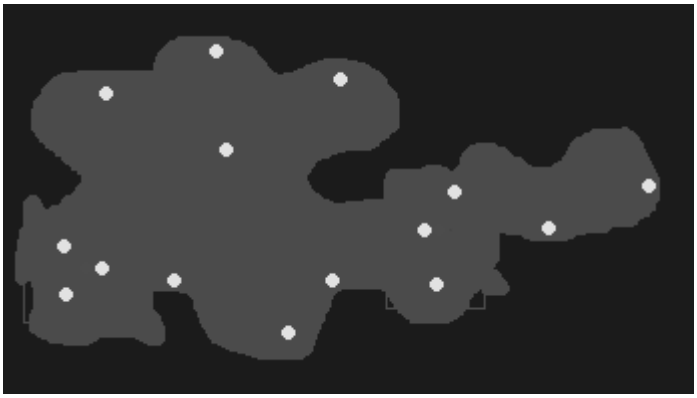
Pojawia się jednak problem skąd brać takie dodatkowe informacje geograficzne do zaawansowanych reguł. Możliwości są trzy. Pierwsza z nich, to wprowadzanie takich informacji ręcznie, co jest pracochłonne, ale mimo wszystko może zostać zrealizowane dla wybranych lotnisk, stacji kolejowych, portów, itd. Druga, to próba ustalenia przynajmniej kontynentu na podstawie pozycji GPS – jest to możliwe, gdy pokryje się kontynenty prostokątami i sprawdzi w sposób analityczny, do którego z nich należy dana pozycja. Trzecią możliwością jest ustalenie nazwy miejscowości, w której znajduje się dana stacja (często występuje ona w nazwie) i wykorzystanie informacji dostępnych w Internecie do uzupełnienia brakujących danych[10] - może to oczywiście zostać wykonane już w momencie dodawania stacji do bazy danych systemu, a nie dopiero w momencie odwołania do niej.

[10] Źródłem takich informacji może być Getty Thesaurus of Geographic Names [36].

Oprócz wspomnianych reguł wpływających na wybór potencjalnych środków transportu jest jeszcze jeden czynnik mający wpływ na ostateczny kształt przygotowywanej listy. Otóż, jako ograniczenie podczas planowania, może zostać podana lista środków transportu, których nie należy brać pod uwagę. Powodem takiej możliwości jest to, że niektóre osoby nie lubią latać, innym zależy na czasie i z góry odrzucają możliwość rejsu statkiem, itd. Z listy przygotowanej przez reguły są usuwane środki zabronione przez użytkownika chyba, że oznaczałoby to usunięcie wszystkich możliwości.

4.2.1.3 Wybór *n* stacji dla danego środka transportu leżących najbliżej danego punktu

Wcześniej zostało stwierdzone, że aby wyznaczyć stacje będące początkiem i końcem głównego etapu wybiera się po prostu pewną ich ilość leżących najbliżej początku i końca odcinka, który jest aktualnie dzielony i następnie rozpatruje wszystkie możliwe pary między nimi. Warto zauważyć, że stwierdzenie, że jakiś etap zaczyna się na pewnej stacji i kończy na drugiej, nie oznacza, iż w drugim etapie planowania będziemy szukać tylko połączeń bezpośrednich między tymi dwoma stacjami. Te dwie wybrane stacje mogą być, na przykład, dwoma lokalnymi stacjami kolejowymi, ale połączenie między nimi będzie się potem składało najpierw z jazdy pociągiem podmiejskim, potem ekspresem i na koniec znowu pociągiem podmiejskim, z przesiadkami na zupełnie innych stacjach. Podobnie będzie wyglądała sprawa w przypadku innych środków transportu. Istotne jest jedynie to, że każde z szukanych połączeń musi się zaczynać i kończyć w punktach początku i końca danego etapu; odwiedzane stacje po drodze i przesiadki w obrębie tego samego środka transportu są dowolne. Widać stąd, że wybór takich stacji jest wystarczający. Stacji wybiera się *n*, gdyż nie ma gwarancji, że takie połączenie kombinowane uda się znaleźć tylko dla dwóch najbliższych. Aby dodatkowo zwiększyć szanse, stosuje się właśnie łączenie we wszystkie możliwe pary – zwiększy to także ilość znalezionych rozwiązań, co da użytkownikowi szerszy wybór.



Rysunek 7 Wybór n stacji w pobliżu początku i końca podróży (n = 3). Czerwone punkty oznaczają początek i koniec dzielonego odcinka podróży, a prostokąty oznaczają wybrane stacje.

4.2.1.4 Dodawanie wiedzy eksperckiej

Dodawanie wiedzy eksperckiej ma na celu podniesienie atrakcyjności przygotowywanych planów podróży. Jest ono realizowane poprzez umożliwienie dodawania dodatkowych par stacji do rozpatrzenia, które powinny zostać zdefiniowane na podstawie położenia początku i końca danego etapu podróży. Mając dane te punkty można, na przykład, stwierdzić, że użytkownik chce jechać z Austrii do Danii. Najbardziej oczywiste plany przebiegałyby mniej więcej po linii prostej, niezależnie, czy byłby to lot samolotem, czy dojazd pociągiem nad Bałtyk i potem rejs promem, a może jazda autokarem. Właśnie też takich rozwiązań szukałby system stosując zasady podziału dążące do minimalizacji długości trasy. Ogólnie jest to słuszne postępowanie, jednak może się okazać, że jest promocja i można bardzo tanio, za symboliczne 1 Euro, polecieć z Anglii do Danii. W takim przypadku opłaca się dojechać do Anglii i stamtąd polecieć samolotem. Reguła ekspercka musiałaby, w tym przypadku, dodać parę lotnisk (na przykład Heathrow i Kastrup).

Problem z zasadami eksperckimi polega na tym, że są one okresowe - często się pojawiają i znikają. Stąd konieczność ich ciągłego uaktualniania. Może to być wykonywane ręcznie lub automatycznie, jeśli dysponuje się odpowiednim źródłem informacji. W obu przypadkach dane te nie mogą zostać ma stałe umieszczone w kodzie programu i muszą być zapisywane i wyciągane z bazy

danych. To z kolei oznacza, że korzystanie z nich będzie kosztowne czasowo i na pewno nie będzie mogło być wykonywane na każdym poziomie rekurencji – stąd ograniczenie maksymalnego poziomu, na którym wiedza ekspercka jest dodawana przez pewną stałą (na przykład równą 2).

4.2.1.5 Usuwanie podziałów znacząco gorszych od innych

Aby dodatkowo polepszyć jakość przygotowywanych planów, można także usunąć z przygotowanej listy podziałów podróży na etapy, te podziały, które są znacząco gorsze od pozostałych. Usunięcie najgorszych podziałów już teraz sprawi, że do drugiego etapu planowania trafią tylko te stosunkowo dobre, co zwiększy szanse uzyskania dobrych jakościowo planów końcowych, a także przyspieszy proces planowania, gdyż w etapie drugim będzie mniej małowartościowych przypadków do rozpatrzenia. Usuwanie podziałów gorszych od innych jednak zawsze zostawia pewną minimalną ilość podziałów, nawet gdyby miały one zawierać takie znacząco gorsze od pozostałych, aby nie ograniczyć zbytnio ilości możliwości przeglądanych w drugim etapie algorytmu.

Aby jednak mówić o dobrych, czy też gorszych podziałach, należy wprowadzić pojęcie miary jakości, przy pomocy której będzie się oceniać uzyskane podziały między sobą. Jako taką miarę można wykorzystać parę czynników (najlepiej wszystkie z nich):

- fakt posiadania niezaplanowanego dystansu na początku lub końcu podróży (na przykład, gdy nie da się dojechać bezpośrednio do żadnej okolicznej stacji)
- długości takich niezaplanowanych dystansów
- oraz ogólną długość zaplanowanej trasy.

Dwa pierwsze kryteria są ze sobą mocno powiązane. Gdy wśród znalezionych planów, są takie, które z obu końców są blisko zarówno początku jak i końca podróży, to wtedy można pousuwać te plany, które nie spełniają tego warunku – lepszy jest plan, który pokazuje drogę już od lokalnej stacji kolejowej, niż dopiero

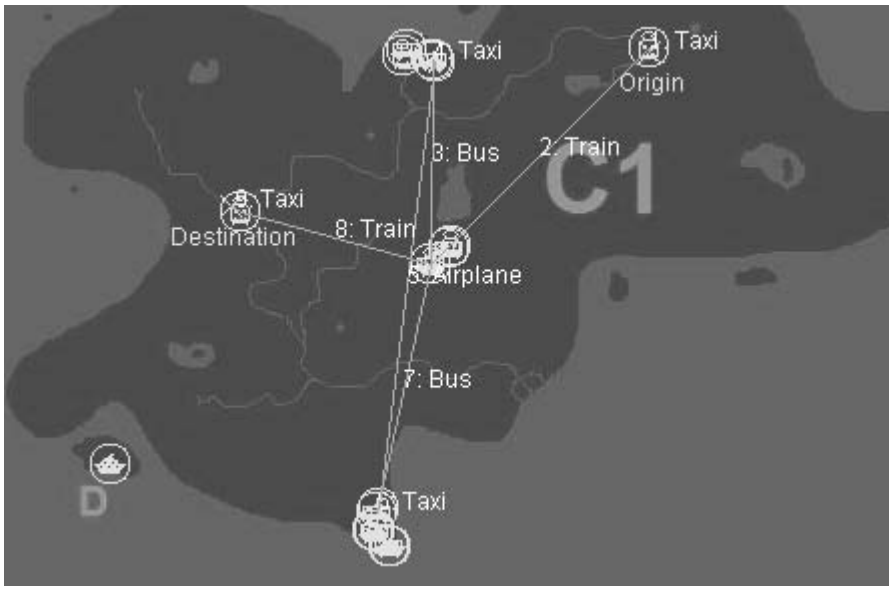
od lotniska położonego 50 km dalej. W przypadku, gdy nie ma planów spełniających ten warunek z obu stron, można usunąć te plany, które go w ogóle nie spełniają pod warunkiem, że są plany spełniające go przynajmniej z jednej strony trasy. Natomiast, gdy dysponujemy tylko planami, które niestety, ale z obu stron zawierają pewne niezaplanowane odcinki dojazdów, to zostawiamy je wychodząc z założenia, że lepsze to niż brak jakiegokolwiek rozwiązania – to, że system nie dysponuje wiedzą, aby ułożyć plan na jakimś lokalnym odcinku, nie znaczy, że użytkownik nie jest w stanie tego zrobić sam, a reszta trasy (ta zaplanowana) i tak będzie dla niego dużym ułatwieniem. Oprócz usuwania bazującego na samym fakcie posiadania niezaplanowanych odcinków, można także przeprowadzić minimalizację długości tych niezaplanowanych odcinków. Chodzi o to, żeby spośród tych planów wybrać te, które mają te niezaplanowane odcinki jak najkrótsze – lepszy jest brak planu dojazdu do stacji kolejowej odległej o 10 km, niż do takiej odległej o 50 km (Rysunek 8 i Rysunek 9). Podobnie jak w poprzednim przypadku, minimalizować można biorąc pod uwagę obie strony trasy lub tylko jedną z nich w zależności od tego czy mamy do czynienia z planami z niezaplanowanymi odcinkami z obu stron, czy też tylko z jednej.

Rysunek 8 Krótszy niezaplanowany odcinek

Rysunek 9 Dłuższy niezaplanowany odcinek

Usuwać można także te plany, które są znacząco dłuższe od reszty. Chodzi tu głównie o usunięcie skrajnych przypadków, będących rezultatem wyboru najbliższych stacji i łączenia ich na zasadzie każdy z każdym – mogą wtedy powstawać plany, które niepotrzebnie zawierają charakterystyczne „zygzaki”

(Rysunek 10). Co prawda, na tym etapie planowania nie znamy jeszcze dokładnych tras przebiegu połączeń między stacjami stanowiącymi początek i koniec poszczególnych etapów, to jednak odległość tych dwóch stacji od siebie po linii prostej ogranicza te długości od dołu – sumując je można ustalić minimalną długość trasy i na jej podstawie dokonać odrzucenia zbyt długich planów. Podczas usuwania zbyt długich planów nie są w ogóle brane pod uwagę te z nich, które zawierają w sobie etapy dodane przez wiedzę ekspercką (te mogą i często są znacznie dłuższe).



Rysunek 10 Przykład zygzaka powstającego podczas generowania podziału podróży na etapy

4.2.2 Etap II – implementacja poszczególnych etapów

W poprzednim rozdziale został przedstawiony sposób generowania podziału podróży na etapy pokonywane różnymi środkami transportu – I etap planowania. Natomiast w tym rozdziale zostanie przedstawiony etap II polegający na implementacji tych etapów już konkretnymi połączeniami. Oczywiście szukana implementacja musi spełniać pewne warunki. Jasnym jest, że należy dążyć do minimalizacji czasów oczekiwania podczas przesiadek oraz do spełnienia dodatkowych ograniczeń narzuconych przez użytkownika. Spełnienie tych warunków nie zawsze będzie możliwe – na przykład, czasami nie będzie możliwe znalezienie takich połączeń, aby czas oczekiwania podczas przesiadek

był nie większy niż maksymalny określony przez użytkownika. W takich przypadkach będą brane pod uwagę możliwości spoza dopuszczalnych, znów wychodząc z założenia, że lepszy jest plan nie do końca spełniający oczekiwania użytkownika niż brak jakiegokolwiek planu. Jednak czas przesiadki to nie jedyne ograniczenie, jakie może się pojawić. Ktoś może wymagać, aby podróż odbywała się tylko pierwszą klasą lub, aby do jedzenia była serwowana kuchnia chińska. Możliwości jest oczywiście więcej, lecz są to tylko ograniczenia drugorzędne i podobnie jak maksymalny czas oczekiwania podczas przesiadek mogą zostać złamane, gdy nie ma innej możliwości. Istnieją też jednak ograniczenia, których nie można złamać pod żadnym pozorem. Do nich można zaliczyć datę wyjazdu lub datę graniczną, na którą chce się dotrzeć, albo też ilość czasu wymaganego na bycie, na przykład, przed odlotem na lotnisku, lub też maksymalną ilość przesiadek. Wydaje się, że to ostatnie ograniczenie mogłoby być również czasami łamane z braku innych możliwości, jednak jest ono ograniczeniem, które w dość znaczący sposób limituje rozmiar przeglądanej przestrzeni możliwości oraz jest swego rodzaju warunkiem stopu dla algorytmu i dlatego zostało przyjęte, że nie będzie możliwości jego łamania. W momencie, gdy dla użytkownika duża ilość przesiadek nie stanowi problemu, to wystarczy to ograniczenie ustawić na dużą wartość. Na samym końcu procesu planowania połączenia niespełniające ograniczeń zostaną usunięte, pod warunkiem, że nie będzie to oznaczało usunięcia wszystkich możliwości (rozdział 4.2.2.7). Zostaną także usunięte połączenia bardzo podobne do siebie (rozdział 4.2.2.8).

Sama implementacja podziałów podróży na etapy będzie się odbywała podział po podziale, każdy niezależnie od reszty. Ponieważ podziałów będzie wiele, a każdy z nich może mieć wiele implementacji, to w rezultacie osiągnie się sporą liczbę rozwiązań, które będą mogły być sortowane według różnych kryteriów (na przykład: czas, koszt lub średni czas oczekiwania na przesiadkę). Dzięki temu proces planowania będzie mógł być przeprowadzony tylko raz, a użytkownik będzie mógł przeglądać możliwości pod różnymi kątami bez konieczności jego powtarzania.

Implementacja na poziomie pojedynczego etapu będzie polegała na wyborze pewnych połączeń startowych dla pierwszego etapu i symulacji jazdy

każdym z nich z możliwością dokonywania przesiadek, aż do osiągnięcia końca etapu. Dla każdego następnego etapu połączeniami startowymi będą te, które będą pasowały czasowo do końca poprzedniego etapu (rozdział 4.2.2.2).

4.2.2.1 Opis działania etapu II algorytmu planowania

Poniżej zostały opisane działania przeprowadzane dla pojedynczego podziału podróży na etapy z listy przygotowanej w etapie I algorytmu planowania. Rozwiązania uzyskane dla każdego z podziałów na etapy są zbierane na wspólnej liście *wszystkich implementacji wszystkich podziałów na etapy*. Lista ta będzie stanowiła zbiór wszystkich planów podróży, które potem będą prezentowane użytkownikowi. Podobnie, jak w przypadku opisu etapu I planowania, w kolejnych rozdziałach zostały opisane pewne szczegóły poniższego opisu. W poniższym opisie pojęcie „data" oznacza datę oraz godzinę. Jest w nim też wykorzystywana pewna struktura nazywana zadaniem, która zostanie teraz opisana, aby ułatwić zrozumienie działania algorytmu. Celem jej użycia jest utrzymywanie informacji o aktualnie przebytej drodze w postaci uporządkowanej listy odcinków podróży – jest to de facto zapis dokonanych wyborów dotyczących przesiadek. Każdy z tych odcinków zawiera informację o jego początku i końcu (plus ewentualnie odpowiadającym im stacjom), użytym środku transportu, odpowiadającym mu połączeniu[11] oraz o datach odjazdu z lokacji początkowej i dotarcia do lokacji końcowej. Daty jednak nie zawsze mogą być ustawione od razu, na przykład, gdy odcinek jest pokonywany pieszo i nie wiadomo jeszcze ile ma się czasu na dotarcie przed odjazdem pociągu z następnego etapu podróży. Brakujące daty są uzupełnianie w końcowej fazie algorytmu – wtedy zdefiniowane są już wszystkie możliwe czasy i można dobrać do nich odpowiednio te jeszcze niezdefiniowane – wtedy się okazuje, czy na ten przykładowy pociąg trzeba iść szybkim krokiem, czy można iść spacerkiem lub nawet zatrzymać się na chwilę w kawiarni. Kolejne odcinki są sukcesywnie dodawane w miarę dokonywania przesiadek podczas przeglądania przestrzeni

[11] Przez połączenie rozumiany jest rozkład jazdy dla wszystkich odwiedzanych stacji wraz z czasami przybycia i odjazdu z każdej z nich dla pewnego środka transportu.

możliwości. Dodatkowo, każde zadanie posiada listę już odwiedzonych stacji, aby uniknąć pętli na trasie podróży. Warto zauważyć, że każdemu etapowi podróży odpowiada przynajmniej jeden odcinek. Jest ich więcej, gdy w ramach danego etapu odbyły się jakieś przesiadki, na przykład, z jednego pociągu do drugiego.

Rozwiązywanie pojedynczego podziału podróży na etapy rozpoczyna się od stworzenia pustej listy *implementacji realizujących już rozwiązane etapy*. Następne kroki będą realizowane kolejno dla każdego etapu z rozpatrywanego podziału podróży na etapy.

Jeśli środek transportu skojarzony z aktualnym etapem jest środkiem transportu, dla którego nie istnieje pojęcie rozkładu jazdy (na przykład taksówka), to tworzony jest nowy odcinek pokrywający cały etap. Data jego rozpoczęcia jest ustawiana na datę zakończenia poprzedniego odcinka. Gdy takiego odcinka nie ma, to jest ona oznaczana do późniejszego ustalenia. Data zakończenie odcinka jest od razu oznaczana do późniejszego ustawienia. Jeśli istnieją implementacje na liście *implementacji realizujących już rozwiązane etapy*, to nowy odcinek jest dodawany do każdej z nich. Po dodaniu usuwane są te implementacje, które łamią limit ilości przesiadek. Jeśli usunięcie takich implementacji usunęło je wszystkie, to należy wyjść z procedury – nie istnieje implementacja etapu, a więc i całego podziału na etapy.

Natomiast, jeśli środek transportu skojarzony z aktualnym etapem jest środkiem transportu, dla którego istnieje pojęcie rozkładu jazdy, to możliwe są dwa przypadki: został już odwiedzony etap z tego typu środkiem transportu i lista *implementacji realizujących już rozwiązane etapy* nie jest pusta lub zachodzi przypadek przeciwny.

Gdy zachodzi przypadek pierwszy, oznacza to, że należy kontynuować już rozwiązane etapy. Tworzone są zadania robocze na podstawie implementacji z listy *implementacji realizujących już rozwiązane etapy*. Dla każdej takiej implementacji brany jest jej czas zakończenia. Dodawany do niego jest czas niezbędny na dokonanie przesiadki. Jeśli ostatni odcinek aktualnej implementacji nie miał ustawionej daty końcowej (był to odcinek pokonywany środkiem

transportu bez pojęcia rozkładu jazdy), to jest rezerwowany czas na jego pokonanie. Data jest także modyfikowana o czas potrzebny na przybycie, na przykład, na lotnisko przed odlotem. Otrzymana data jest datą, od której można wyszukiwać połączeń dla aktualnego środka transportu (rozdział 4.2.2.3). Jeśli nie ma połączeń spełniających ograniczenia przesiadki, to brana jest pewna ilość z najbliższych odchodzących połączeń. Dla każdego wybranego połączenia jest tworzone zadanie robocze, które jest dodawane do menadżera zadań pod warunkiem, że nie łamie ono ograniczenia maksymalnej ilości przesiadek.

Gdy zachodzi przypadek przeciwny, oznacza to, że istnieje konieczność dokonania inicjacji połączeń, gdyż do tej pory były odwiedzane tylko etapy ze środkami transportu bez pojęcia rozkładu jazdy lub nie były odwiedzana żadne etapy i nie została jeszcze ustalona żadna konkretna data odjazdu (wszystkie są na razie oznaczone do późniejszego wyznaczenia). Aby dokonać inicjacji, jako datę wyjazdu przyjmuje się datę graniczną rozpoczęcia podróży podaną przez użytkownika. W analogiczny sposób jak poprzednio jest ona przesuwana (o czas przesiadek, rezerwacji i wcześniejszego bycia na stacji). Otrzymana data jest datą, od której można wybrać pewną ilość odchodzących połączeń z aktualnej stacji. Dla wybranych połączeń znowu tworzone są zadania robocze, które są dodawane do menadżera zadań, pod warunkiem, że nie łamią limitu ilości przesiadek.

Następnie przystępuje się do rozwiązywania zadań przygotowanych w obu poprzednich przypadkach (rozdział 4.2.2.2). Na początku tworzona jest lista *implementacji realizujących aktualny etap*. Zadania są rozwiązywane tak długo, jak długo są one dostępne w menadżerze zadań. Brane jest kolejne zadanie z menadżera zadań. Wyznacza się dla niego kolejną stację w jego aktualnym połączeniu (jeśli już nie ma kolejnej stacji, to przechodzi się do kolejnego zdania). Jeśli była ona już odwiedzona w tym zadaniu, to przechodzi się do kolejnego zadania, ignorując aktualne zadanie (inaczej powstałby cykl). Następnie dokonuje przejścia się do tej stacji. Jeśli jest to stacja końcowa, to należy utworzyć nowy odcinek od końca ostatniego odcinka (lub początku etapu, gdy takiego nie ma) do aktualnej stacji. Nowy odcinek jest dodawany do zadania, a samo zadanie jest umieszczane na liście *implementacji realizujących aktualny etap*, po czym przechodzi się do kolejnego zadania. Gdy jednak nie był to koniec etapu, to

sprawdzane jest, czy nie został już w tym zadaniu osiągnięty limit ilości przesiadek. Jeśli nie, to generowana jest lista możliwych przesiadek dla aktualnej stacji. Dla każdej przesiadki jest tworzone odrębne zadanie, które jest umieszczane w menadżerze zadań pod warunkiem, że reguły odrzucające przesiadki nie uznają, że nie warto jej rozpatrywać. Na koniec aktualne zadanie jest zwracane do menadżera zadań do dalszego rozpatrywania.

Po rozpatrzeniu wszystkich zadań dokonuje się zamiany listy *implementacji realizujących już rozwiązane etapy* listą *implementacji realizujących aktualny etap*. Jeśli lista ta jest pusta, to należy wyjść z procedury – nie istnieje implementacja etapu, a więc i całego podziału na etapy. Następnie przystępuje się do ustalenia wszystkich brakujących dat oraz poprawienia dat odcinków pokonywanych środkami transportu, dla których nie istnieje pojęcie rozkładu jazdy różnych niż spacer (rozdział 4.2.2.5). Na koniec usuwa się implementacja łamiące ograniczenia (rozdział 4.2.2.7).

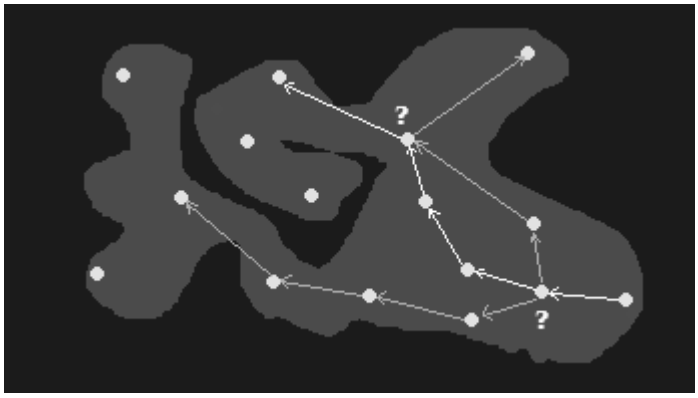
Po rozwiązaniu wszystkich podziałów na etapy z listy *wszystkich implementacji wszystkich podziałów na etapy* usuwa się implementacje podobne do siebie (rozdział 4.2.2.8).

4.2.2.2 Implementacja połączeniami pojedynczego etapu podróży

Jak już to zostało powiedziane we wstępie i w dokładnym opisie II etapu algorytmu planowania, etapy są implementowane jeden po drugim. Rozwiązania poprzedniego są kontynuowane w kolejnym etapie. W ten sposób, po rozwiązaniu ostatniego etapu, jego rozwiązania będą planem całej podróży. Każdy z przygotowanych wcześniej podziałów może wygenerować pewną ilość rozwiązań. Ich suma stanowi zbiór wszystkich możliwości dotarcia z początku do końca podróży.

Na poziomie pojedynczego etapu rozwiązania są poszukiwane na zasadzie odwiedzania wszystkich kolejnych stacji na trasie aktualnego połączenia i dokonywania ewentualnych przesiadek, z których każda jest rozpatrywana

oddzielnie - jest to jakby symulowanie jazdy przez pasażera i obserwowanie, co dają różne możliwości transportowe. Możliwe przesiadki są odkładane do rozwiązania na potem jako zadania. Kolejnością ich rozpatrywania zajmuje się menadżer zadań, który działa na zasadzie stosu, aby maksymalnie ograniczyć ilość pamiętanych na raz zadań. Oczywiście nie są dokonywane wszystkie możliwe przesiadki, gdyż to prowadziłoby do przeglądania olbrzymiej ilości możliwości. Pod uwagę są brane tylko te, które po pierwsze są zgodne z narzuconymi przez użytkownika ograniczeniami (z pewnymi wyjątkami – rozdział 4.2.2.3), a po drugie te, które dodatkowo nie zostały odrzucone przez specjalne reguły doboru przesiadek (rozdział 4.2.2.4). Ich zastosowanie umożliwia jeszcze lepsze przycięcie drzewa możliwości i w rezultacie powoduje, że algorytm działa szybciej.



Rysunek 11 Przykład przechodzenia połączeń stacja po stacji

Innym sposobem na przyspieszenie wyszukiwania jest trzymanie się połączeń bezpośrednich. Oznacza to, że jeśli istnieje możliwość dokonania przesiadki na połączenie, którym da się bezpośrednio dojechać do celu etapu, to wtedy przesiadki są możliwe tylko na połączenia, którymi można tam dotrzeć szybciej. Nie są już wtedy rozpatrywane inne możliwości. Zasada ta dotyczy tylko pojedynczych zadań, więc nie oznacza to, że jeśli jedno z nich będzie miało taką możliwość, to przy innych przestaną być rozpatrywana połączenia niebezpośrednie. Taka zasada przyspiesza wyszukiwanie z reguły w końcowej fazie przeszukiwania, aczkolwiek może i dużo wcześniej, jeśli stacja docelowa jest ważniejszym węzłem komunikacyjnym i posiada połączenia bezpośrednie z odległymi miejscami.

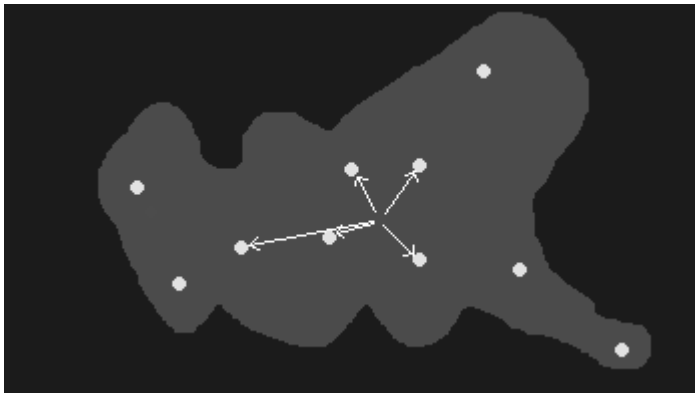
Zostało już powiedziane, że rozwiązania jednego etapu są punktami wyjścia dla kolejnego. Jednak rodzi się pytanie, co z pierwszym etapem? Jak zainicjować cały proces? Sposób inicjacji pierwszych połączeń został opisany w następnym podrozdziale (4.2.2.3). Tutaj zostanie przedstawiony moment, w którym jest on wykonywany. Wbrew pozorom nie musi to być pierwszy etap. Do takiej sytuacji dojdzie, gdy pierwszy etap jest pokonywany, na przykład, pieszo lub taksówką, a ogólniej mówiąc, środkiem transportu, w którym nie istnieje pojęcie rozkładu jazdy. Wybór połączeń początkowych odbywa się po napotkaniu pierwszego etapu ze środkiem transportu, dla którego istnieją rozkłady jazdy, ponieważ to do niego trzeba dopasowywać w sposób elastyczny pozostałe fragmenty. Dla wcześniejszych etapów jest rezerwowany pewien minimalny czas potrzebny na ich pokonanie i przesiadki, jednak przypisanie im dokładnych ram czasowych następuje później (rozdział 4.2.2.5), gdy są już znane granice czasowe następujących po nich etapów. Do rezerwacji czasu dochodzi także, gdy etapy ze środkami transportu bez rozkładów jazdy są napotykane później.

Oprócz rezerwacji czasu na etapy, których czas trwania nie jest znany z góry, konieczne jest także uwzględnianie pewnych przerw na dokonywanie przesiadek (ich długość ogranicza użytkownik) oraz rezerwowanie dodatkowego czasu przed lotem samolotem lub rejsem statkiem na odprawę, która trwa znacznie dłużej niż normalna przesiadka (czas na odprawę jest dodawany oprócz zwykłego czasu na przesiadkę).

4.2.2.3 Wybór połączeń startowych i alternatywnych przesiadek

Aby wybrać wspomniane w poprzednim rozdziale połączenia początkowe, wystarczy wybrać pewną ilość połączeń, które odchodzą jak najszybciej po zadanej dacie wyjazdu. Należy jednak nie brać dowolnych najbliższych czasowo połączeń, lecz dodatkowo wybrać je tak, aby prowadziły one w możliwie jak najwięcej różnych miejsc, aby przypadkowo nie wybrać tylko połączeń, które posiadają taką samą trasę (a właściwie jej początek). Stąd pomysł, aby oprócz samej różnicy czasu od daty wyruszenia, brać także pod uwagę kolejną stację w

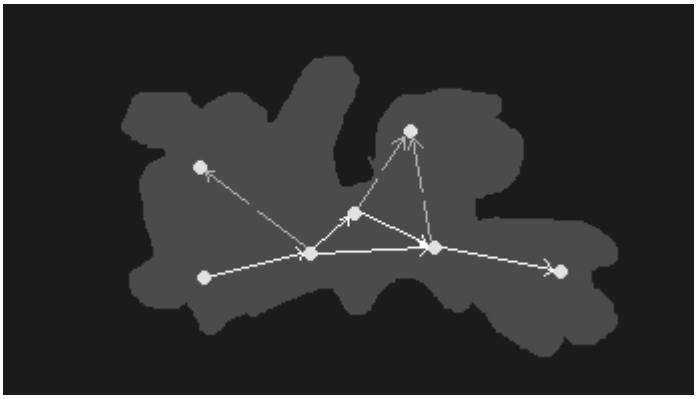
danym połączeniu. Właśnie kolejna stacja jest głównym czynnikiem wyboru początkowych połączeń. Ponieważ jednak musimy wybrać ich skończoną ilość a nie dowolnie dużo, to jako dodatkowe kryterium wyboru będziemy stosować różnicę czasu między odjazdem i datą wyjazdu zadaną przez użytkownika. Podobnie, gdy następne stacje są takie same, to bierzemy to połączenie, które wyrusza szybciej. Stosując te zasady, otrzymamy ograniczoną listę połączeń, z których każde posiada inną kolejną odwiedzaną stację. A dla każdej z tych stacji będziemy mieli wybrane to połączenie, które odchodzi najwcześniej ze stacji początkowej. Ta sama zasada doboru jest także stosowana podczas wyboru alternatywnych przesiadek, czyli w momencie, gdy nie ma przesiadek spełniających ograniczenia podane przez użytkownika (wtedy wybierane są inne w myśl zasady, że lepsze to niż nic). To, co ta zasada daje, to ograniczenie ilości rozpatrywanych przypadków. Kryterium doboru połączeń pozwala wybrać te najlepsze z możliwych przy zachowaniu maksymalnie dużej ich różnorodności, a co za tym idzie nie ograniczanie się do de facto jednego przypadku, jaki mógłby powstać, gdyby były brane połączenia tylko z uwzględnieniem kryterium czasowego – wystarczy jako przykład podać sytuację, gdy z jakiejś stacji kolejowej odchodzą pociągi regionalne co 15 min, a pociągi dalekobieżne co parę godzin (wtedy, w największej ilości przypadków, zostałyby wybrane tylko pociągi regionalne, a tak naprawdę każdy z nich oznacza tą samą trasę i tylko jeden z nich dotrze do celu najszybciej). Podczas wyboru alternatywnych przesiadek są brane pod uwagę także wspomniane już wcześniej połączenia bezpośrednie.



Rysunek 12 Wybór połączeń startowych - każde odchodzi do innej stacji

4.2.2.4 Reguły odrzucania potencjalnych przesiadek

Aby możliwie maksymalnie ograniczyć ilość rozpatrywanych przesiadek (a co za tym idzie ilość przeglądanych możliwości), stosowane są pewne reguły, których celem jest odrzucenie przesiadek, które na pewno nie prowadzą do szukanego celu. Reguły te biorą pod uwagę różne wiadomości, aby zdecydować o przydatności poszczególnych przesiadek. Mogą to być reguły bazujące na ogólnym kierunku połączenia, czyli czy jechanie nim prowadzi nas do celu, czy wręcz przeciwnie – oddala nas. Mogą też sprawdzać czy dane połączenie jest, na przykład, połączeniem lokalnym i nie warto się w nie przesiadać, gdy jest się jeszcze daleko od celu podróży. Reguły te wymuszają także trzymanie się przesiadek bezpośrednich. Mogą one równocześnie odrzucać przesiadki, które nie spełniają ograniczeń narzuconych przez użytkownika, na przykład, gdy skład pociągu nie zawiera pierwszej klasy.



Rysunek 13 Działanie reguł odrzucających przesiadki

Reguły te nie są jednak stosowane podczas wyboru wspomnianych już wcześniej połączeń startowych i alternatywnych przesiadek. Jest to podyktowane chęcią zapewnienia jak największej różnorodności początkowych możliwości przygotowywanych do późniejszego rozpatrzenia. Jednorazowo dopuszcza się możliwości, które normalnie mogłyby nie być dopuszczone przez reguły. Połączenia, które zostały dopuszczone, a nie były korzystne i tak zostaną w następnych krokach wyeliminowane przez reguły, gdyż uniemożliwią one dokonywanie z nich przesiadek i w efekcie nie prowadzą do powstania większych drzew przeszukiwania. Możliwości, które zostaną jednak rozpatrzone mogą

zaowocować wyszukaniem połączeń, które normalnie nie byłyby sprawdzone – w ten sposób tanim kosztem można osiągnąć lepsze rezultaty.

4.2.2.5 Rezerwacja i przypisywanie czasów etapom pokonywanych środkami transportu bez rozkładów jazdy

W rozdziale zajmującym się rozwiązywaniem pojedynczego etapu podróży (4.2.2.2) znalazło się stwierdzenie, że czasy dla etapów, które są pokonywane środkami transportu, dla których nie istnieje pojęcie rozkładu jazdy, nie mają najpierw ustawianych dat definiujących czas ich trwania, lecz jest to robione dopiero na pod koniec działania algorytmu. Jednak, aby móc planować dalej, jest rezerwowany na nie pewien czas – jego ilość jest zależna od środka transportu i liniowo zależy od odległości do pokonania. Rezerwacja jest dokonywana po to, aby wiedzieć, które z połączeń w następnym etapie można brać pod uwagę (połączenia odchodzące po której godzinie będą dobre?). Na koniec, gdy wszystkie połączenie są już ustalone, ustawiane są daty wszystkim takim odcinkom bez dat. Jest to możliwe, gdyż zarówno odcinek poprzedni, jak i odcinek następny mają takie daty przypisane i można je po prostu przepisać. Tutaj widać, jak ważne jest ile tego czasu zostało wcześniej zarezerwowane. Gdyby było tego czasu za mało, to w tym momencie mogłoby się okazać, że, na przykład, na przejście pewnego odcinka pieszo jest go zdecydowanie za mało i w rezultacie cały plan jest niewykonalny. Rezerwacje czasu muszą być dokonywane bardzo ostrożnie. Może się także zdarzyć, że nie istnieje odcinek poprzedni albo następny. Będzie tak, gdy odcinek bez daty jest odcinkiem pierwszym lub ostatnim. Wtedy można przepisać jedną z danych dat, a drugą dobrać w odpowiedni sposób, aby odcinek był odpowiednio długi i nie łamał ograniczeń czasowych narzuconych na cały plan.

W pewnych przypadkach może się okazać, że niewystarczające jest po prostu przepisanie dat z odcinków sąsiednich. Tak na prawdę, można tak postępować tylko, gdy odcinek jest pokonywany pieszo. W każdym innym przypadku (przykładowo taksówka) należy jeszcze z obu końców trwania odcinka

zostawić trochę czasu na dokonanie przesiadki – czas i na to powinien być wcześniej zarezerwowany.

Na koniec warto zauważyć, że odcinki po przypisaniu im dat będą trwały nie krócej niż zarezerwowany na nie wcześniej czas. Oczywiście może się okazać, że będą trwały znacznie dłużej. Jest tak, ponieważ do już zarezerwowanego czasu dochodzi jeszcze dodatkowy narzut w postaci różnicy czasu między końcem rezerwacji a odjazdem pierwszego połączenia po tym końcu.

4.2.2.6 Przypadek implementacji z daną datą graniczną na dojechanie

Przypadek planowania z daną datą limitującą czas dojazdu można rozwiązać w bardzo podobny sposób do przypadku planowania z daną datą wyjazdu. Aby go rozwiązać i dojechać w jak najbliższym momencie przed danym limitem czasowym, należy niejako planować od końca. Okazuje się, że można nawet zastosować dokładnie ten sam algorytm planujący tylko, że należy na samym początku odpowiednio przygotować podziały na etapy – należy je odwrócić. Podczas samego działania algorytmu należy już tylko uważać na operacje na czasie (przesuwanie następuje w drugą stronę) i na fakt, że odwróceniu ulegają pojęcia następnej i poprzedniej stacji. Gdy algorytm zakończy już swoje działanie, trzeba jeszcze z powrotem odwrócić znalezione rozwiązania, gdyż rozwiązują one odwrócony podział na etapy.

4.2.2.7 Usuwanie implementacji, które nie zachowują ograniczeń

Po zebraniu wszystkich możliwych implementacji dla wszystkich podziałów podróży na etapy należy sprawdzić, czy wśród nich znajdują się rozwiązania łamiące narzucone ograniczenia. Mogą one istnieć, gdyż przy wyborze przesiadek przy braku możliwości spełniających ograniczenie maksymalnego czasu przesiadki brane są, o ile istnieją, możliwości, które spowodują przekroczenie tego czasu (łamane mogą być też inne ograniczenia drugorzędne). Jest to podyktowane podejściem, że lepiej mieć rozwiązanie

niedoskonałe niż nie mieć żadnego. W momencie, gdy są już zebrane wszystkie rozwiązania, można sprawdzić, czy wszystkie one łamią zadane ograniczenia – wcześniej takiej możliwości nie było, gdyż etapy były rozwiązywane niezależnie. Jeśli nie, to można usunąć te, które łamią, gdyż zostaną inne, lepsze. Gdy wszystkie osiągnięte rozwiązania łamią ograniczenia, to nie usuwa się ich i pozostawia je wszystkie niezmienione, inaczej nie pozostałyby żadne rozwiązania.

4.2.2.8 Usuwanie podobnych implementacji

Po usunięciu rozwiązań łamiących ograniczenia, należy jeszcze dodatkowo usunąć rozwiązania, które są do siebie podobne. Podobieństwo oznacza w tym przypadku, że oba rozwiązania posiadają taką samą ilość odcinków: odpowiadające sobie odcinki łączą te same stacje i są pokonywane tymi samymi środkami transportu, a także posiadają taką samą datę rozpoczęcia i zakończenia całej podróży. Jedynymi różnicami mogą być czasy pośrednie, to jest czasy dotarcia lub odjazdu ze stacji pośrednich (czyli wykorzystanie różnych połączeń na poszczególnych odcinkach). Plany zawierające tylko takie różnice są sobie tożsame i można usunąć wszystkie oprócz jednego z nich bez straty ogólnej jakości i różnorodności znalezionych planów. Ze wszystkich podobnych do siebie implementacji pozostawiana jest ta, która ma najkrótszy średni czas przesiadki, a w przypadku, gdy takie średnie są sobie równe dla dwóch implementacji, to wtedy decyduje mniejszy maksymalny czas przesiadki.

4.3 Porównanie zaprezentowanego rozwiązania z innymi systemami

Zaproponowane rozwiązanie charakteryzuje się dużymi możliwościami. Potrafi zaplanować podróż między dwoma dowolnymi punktami na świecie bez względu na odległość, jaka je dzieli. Możliwości takiej nie posiadał *Travel Assistant* wykorzystujący wcześniej wspomniane podsystemy: *Theseus'a* i *Heracles'a*, który to mógł planować podróż tylko z wykorzystaniem lotu

samolotem i dojazdów na lotniska prywatnym samochodem lub taksówką, co drastycznie ograniczało jego przydatność. Ograniczało to również obszary, na których był on w stanie działać – wymagane było lotnisko w pobliżu zarówno początku, jak i końca podróży. Zaproponowane rozwiązanie nie posiada takich ograniczeń, a co więcej, pozwala na wykorzystanie wielu środków transportu podczas procesu planowania podróży.

Jego kolejną przewagą jest to, że w momencie, gdy nie jest w stanie zaplanować podróży dokładnie od wyznaczonego początku (lub dokładnie do wyznaczonego końca) podróży, na przykład, gdy nie ma żadnych stacji w okolicy tych punktów, to przygotowuje ono plan zaczynający (kończący) się możliwie najbliżej tych punktów. Gwarantuje to, iż użytkownik nie zostanie pozostawiony bez rozwiązania w przypadku braku odpowiednich danych w systemie.

Elastyczność przedstawionego rozwiązania przejawia się, między innymi, w fakcie, iż umożliwia ono łamanie pewnych ograniczeń nakładanych przez użytkownika w momencie, gdy ich spełnienie nie jest możliwe. Dzięki temu użytkownik otrzymuje pewne propozycje odbycia podróży, podczas, gdy w przypadku innych systemów zostałby poinformowany o braku możliwości spełniających jego ograniczenia. Użytkownik w takiej sytuacji po pierwsze pozostaje bez rozwiązania swojego problemu, a po drugie nie wie, które z narzuconych przez niego ograniczeń uniemożliwiło odnalezienie jakichkolwiek planów podróży – może się okazać, że dopiero po paru próbach rozluźnienia ograniczeń uda mu się odnaleźć pewne rozwiązanie, co oznacza dla niego niepotrzebną stratę czasu.

Wspomniany już *Travel Assistant* posiada także inne ograniczenie – jego proces planowania jest skoncentrowany na celu, który wynika z ograniczeń narzuconych przez użytkownika. Na przykład, gdy użytkownik chce znaleźć najtańsze połączenie, wyszukiwane jest tylko jedno, właśnie najtańsze połączenie; wszystkie inne są odrzucane. W momencie, gdy użytkownik chce zmienić część ograniczeń, konieczne jest powtórzenie części lub całości procesu planowania. W przedstawionym rozwiązaniu takiej konieczności nie ma, gdyż przygotowuje ono od razu całą listę propozycji, które mogą być dowolnie sortowane, co umożliwia

użytkownikowi dokładniejszy przegląd istniejących możliwości odbycia podróży bez konieczności powtarzania samego procesu planowania.

Travel Assistant wymaga również przeglądania praktycznie wszystkich możliwości, aby odnaleźć najlepszy plan. Podobnie jest w przypadku planowania z wykorzystaniem *Hierarchicznych Sieci Zadań*. Przedstawione rozwiązanie posiada mechanizmy reguł ograniczających przeszukiwaną przestrzeń, co umożliwia uniknięcie takiej sytuacji. Co więcej, reguły te mogą być edytowane – daje to możliwość ich łatwego udoskonalania. W przypadku *Hierarchicznych Sieci Zadań* pewne możliwości działań tego typu dają predykaty dzielenia zadań na podzadania, aczkolwiek nie jest to rozwiązanie równie elastyczne co zaproponowane; trudniej je także modyfikować.

Zaproponowane rozwiązanie posiada także inną możliwość, której nie posiadają inne wspomniane systemy – pierwszy etap planowania podróży może zostać wspomożony poprzez specjalny moduł ekspercki, który dostarcza par stacji, które należy uwzględnić podczas planowania. Umożliwia to zmuszenie algorytmu do poprowadzenia trasy właśnie przez te stacje, co może pozwolić na zwiększenie jakości przygotowywanych planów – przypadki normalnie ignorowane lub nie rozpatrywane zostaną uwzględnione.

Zaproponowane rozwiązanie nie posiada jednak pewnych funkcjonalności udostępnianych przez *Travel Assistant*. Po pierwsze, nie jest w stanie przewidywać opóźnień połączeń – zawsze zakłada, że będą one zgodnie z planem. Może to powodować, że plan nie będzie wykonalny na skutek zaistniałego opóźnienia, które można było łatwo przewidzieć, na przykład, gdy dotyczyło to lotów w okresie Bożego Narodzenia lub jazdy autobusem miejski w godzinach szczytu. Po drugie, nie posiada ono możliwości przygotowania od razu planu podróży w dwie strony wraz z pobytem na miejscu (zaproponowaniem hotelu). Konieczne jest dwukrotne zaplanowanie trasy – raz z limitem dojazdu na określony czas i drugi raz – podróży powrotnej – z określonym czasem rozpoczęcia podróży (koniec pobytu). Hotel może zostać zaplanowany na podstawie dat wynikających z obu podróży. Hotel może zostać także wyznaczony przed rozpoczęciem planowania podróży i wtedy istnieje możliwość

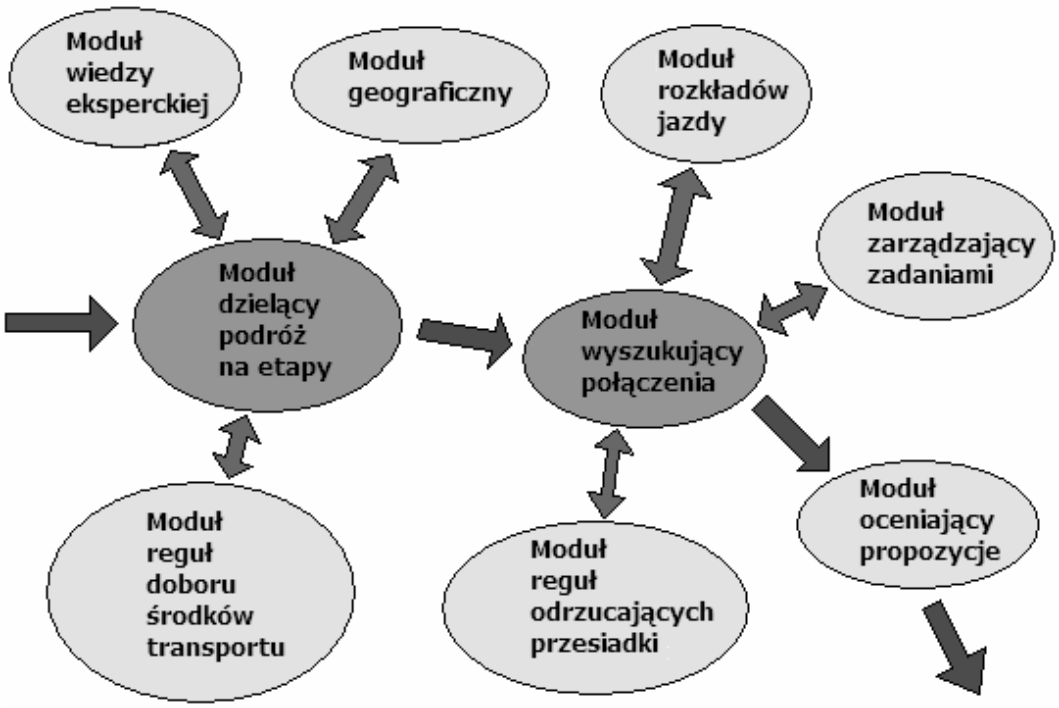
zaplanowania tras prowadzących dokładnie do wybranego hotelu. Po trzecie, *Travel Assistant* posiada już funkcjonalność nadzorowania odbywającej się podróży.

4.4 Implementacja algorytmu planowania

W niniejszym rozdziale zostaną opisane wybrane kwestie związane z implementacją opisanego wcześniej algorytmu. Jako język programowania dla całego wspólnego projektu została wybrana Java. Kod był tworzony przy pomocy platformy Eclipse [35]. Moduły eksperckie bazujące na regułach *jeśli-to* zostały zaimplementowane z wykorzystaniem biblioteki JESS (*Java Expert System Shell*) będącej w chwili obecnej standardem implementacji systemów regułowych w języku JAVA stworzonej przez Ernesta Friedman'a-Hill'a ze Sandia National Laboratories [39].

4.4.1 Architektura zaproponowanego rozwiązania

Implementacja wcześniej przedstawionego algorytmu została zaplanowana w sposób, który uwzględnia modułowość całego rozwiązania. Celem takiego podejście jest stworzenie możliwości łatwego rozszerzenia lub wymienienia poszczególnych komponentów na inne, możliwe, że posiadające znacznie większe możliwości. Jest to możliwe dzięki zdefiniowaniu interfejsów, które muszą być implementowane przez odpowiednie moduły. Schemat architektury przygotowanej implementacji znajduje się poniżej (Rysunek 14). Na potrzeby tej pracy zostały stworzone uproszczone moduły: geograficzny i rozkładów jazdy. Moduły reguł doboru środków transportu, reguł odrzucających przesiadki oraz wiedzy eksperckiej posiadają pełną funkcjonalność, aczkolwiek można do nich dodawać dodatkowe reguły nieprzewidziane przez autora tej pracy. Moduł zarządzający zadaniami i oceniający propozycje posiadają pełną funkcjonalność. Dokładniejszy opis wszystkich modułów znajduje się w kolejnych rozdziałach.



Rysunek 14 Schemat systemu planowania podróży

Na widocznym schemacie strzałki czerwone pokazują typowy przepływ danych w systemie planującym. Cały proces zaczyna się od podania dwóch punktów, między którymi należy zaplanować podróż. Są one przekazywane do modułu dzielącego podróż na etapy. Generuje on listę możliwych podziałów. Przygotowane podziały są następnie przekazywane do modułu wyszukującego połączenie, który dokonuje implementacji podziałów z listy konkretnymi połączeniami. Wyznaczone przez niego implementacje są już rozwiązaniami – gotowymi planami podróży, jednak przed zaprezentowaniem ich użytkownikowi są one dodatkowo ocenianie pod kątem preferencji użytkownika na podstawie danych historycznych o jego podróżach, co umożliwia ich posortowanie i wyświetlenie na pierwszym miejscu tych potencjalnie najbardziej atrakcyjnych.

Strzałki niebieskie oznaczają natomiast interakcję głównych modułów planujących z modułami pomocniczymi, które są odpowiedzialne za dostarczanie odpowiednich danych lub podejmowanie pewnych dodatkowych decyzji, które mogą przyspieszyć proces planowania.

4.4.1.1 Moduł dzielący podróż na etapy

Zadaniem tego modułu, jak wskazuje jego nazwa, jest realizacja I etapu przedstawionego wcześniej algorytmu planowania podróży. Na wejściu otrzymuje on dwa punkty. Jego zadaniem jest wygenerowanie listy podziałów podróży między tymi dwoma punktami na etapy pokonywane różnymi środkami transportu. Do realizacji tego zadania wykorzystywane jest wsparcie trzech innych modułów:

- Wiedzy eksperckiej
- Geograficznego
- Reguł doboru środków transportu

Lista podziałów podróży na etapy stanowiąca wyjście tego modułu jest przekazywana dalej do modułu wyszukującego połączenia, który będzie wyszukiwał odpowiednich połączeń implementujących podziały z tej listy.

4.4.1.2 Moduł wiedzy eksperckiej

Zadaniem modułu wiedzy eksperckiej jest dostarczanie modułowi dzielącemu podróż na etapy pewnych dodatkowych par stacji w zależności od aktualnie dzielonego odcinka podróży. Pary te umożliwiają zmuszenie algorytmu planującego podróż do rozpatrywania możliwości, które normalnie nie zostałyby wzięte pod uwagę. Mogą to być, na przykład, stacje, między którymi są połączenia w promocyjnych cenach lub stacje, które pomagają algorytmowi w pewnych sytuacjach, w których nie radzi on sobie najlepiej. Moduł dzielący podróż na etapy może funkcjonować bez tego modułu.

Definicja odpowiedniego interfejsu znajduje się w Załączniku III.

4.4.1.3 Moduł geograficzny

Moduł geograficzny również współpracuje z modułem dzielącym podróż na etapy. Jego zadaniem jest dostarczanie list stacji dla danego środka transportu leżących w podanym prostokątnym obszarze. Dzięki tej funkcjonalności możliwy jest wybór stacji leżących najbliżej pewnego punktu (często innej stacji). Bez tego modułu moduł dzielący podróż na etapy nie byłby w stanie funkcjonować, gdyż nie posiadałby zdolności łączenia ze sobą różnych środków transportu ze względu na fakt, iż nie mógłby stwierdzać, które stacje znajdują się blisko siebie i jak przedostawać się z jednej na drugą.

Definicja odpowiedniego interfejsu znajduje się w Załączniku IV.

4.4.1.4 Moduł reguł doboru środków transportu

Moduł reguł doboru środków transportu jest ostatnim z modułów współpracujących z modułem dzielącym podróż na etapy. Podobnie jak moduł wiedzy eksperckiej, nie jest on niezbędny do działania modułu dzielącego. Jednak jego wykorzystanie pozwala na ograniczanie ilości rozpatrywanych możliwości w etapie II algorytmu planującego podróż. Jego zadaniem jest dostarczenie listy potencjalnych środków transportu, których można użyć do pokonania odcinka między dwoma danymi punktami. Reguły pozwalają na wykluczenie części ze środków transportu i dzięki temu możliwe jest ograniczenie ilości możliwości do późniejszego rozpatrzenia.

Definicja odpowiedniego interfejsu znajduje się w Załączniku V.

Moduł ten został zaimplementowany przy wykorzystaniu biblioteki JESS (*Java Expert System Shell*), co umożliwia dynamiczną edycję wykorzystywanych reguł. Zapis wcześniej opisanych reguł w języku JESS znajduje się w Załączniku IX.

4.4.1.5 Moduł wyszukujący połączenia

Moduł ten otrzymuje na wejściu listę podziałów podróży na etapy. Dla każdego takiego podziału wyszukuje on połączeń, które będą go implementowały przy zachowaniu ograniczeń narzuconych przez użytkownika. Podczas realizacji swoich zadań moduł ten współpracuje z następującymi modułami:

- Rozkładów jazdy
- Zarządzającym zadaniami
- Reguł odrzucających przesiadki

Wynikiem działania tego moduły będzie lista znalezionych rozwiązań problemu podróży między dwoma zadanymi punktami. Lista ta będzie mogła być przeglądana i sortowana według różnych kryteriów przez użytkownika.

4.4.1.6 Moduł rozkładów jazdy

Zadaniem tego modułu jest dostarczanie list połączeń, które posiadają zadaną stację na swojej liście stacji leżących na trasie danego połączenia. Moduł ten jest pośrednikiem między bazą wszystkich połączeń zdefiniowanych w „świecie" a modułem wyszukującym połączeń. Nic nie stoi także na przeszkodzie, aby był on odpowiedzialny za utrzymywanie listy wszystkich połączeń. Funkcjonalność udostępniania przez ten moduł jest niezbędna do funkcjonowania modułowi wyszukującemu połączeń.

Definicja odpowiedniego interfejsu znajduje się w Załączniku VI.

4.4.1.7 Moduł zarządzający zadaniami

Moduł ten również jest niezbędny do działania modułowi wyszukującemu połączeń. Jest on przez niego wykorzystywany do pamiętania i zarządzania kolejnością rozpatrywania zadań podczas procesu wyszukiwania połączeń dla

podziałów podróży na etapy. Wskazane jest, aby działał on na zasadzie stosu, gdyż ogranicza to ilość pamiętanych przez niego na raz zadań. Moduł ten mógłby być również odpowiedzialny za nadzór nad równoległym rozpatrywaniem zadań.

4.4.1.8 Moduł reguł odrzucających przesiadki

Natomiast moduł reguł odrzucających przesiadki nie jest już niezbędny do funkcjonowania modułu wyszukiwania połączeń. Jednak jego wykorzystanie umożliwia w znaczący sposób ograniczyć ilość rozpatrywanych możliwości. Jego zdanie polega na odrzuceniu z listy możliwych przesiadek tych, które nie są wartościowe.

Definicja odpowiedniego interfejsu znajduje się w Załączniku VII.

Moduł ten został zaimplementowany przy wykorzystaniu biblioteki JESS (*Java Expert System Shell*), co umożliwia dynamiczną edycję wykorzystywanych reguł. Zapis wcześniej opisanych reguł w języku JESS znajduje się w Załączniku X.

4.4.1.9 Moduł oceniający propozycje

Przygotowane przez moduł wyszukiwania połączeń propozycje podróży mogą zostać ocenione przez ten moduł pod kątem podobieństwa do poprzednich propozycji podróży, które zostały przez użytkownika albo zaakceptowane, albo odrzucone (jawnie wskazane jako nieodpowiednie). Propozycje najbardziej podobne do tych zaakceptowanych powinny najbardziej odpowiadać użytkownikowi, zaś te podobne do odrzuconych propozycji w naturalny sposób będą mu mniej odpowiadały. Bazując na podobieństwie do historycznych danych można posortować propozycje od tych najbardziej odpowiednich do tych najmniej. Potencjalnie najlepsze propozycje znajdą się na pierwszym miejscu wśród propozycji prezentowanych użytkownikowi.

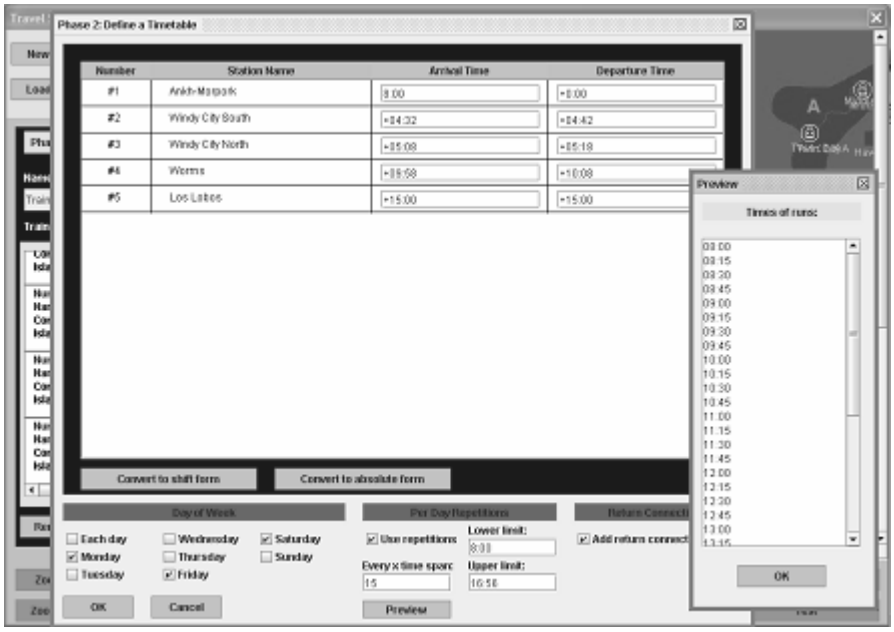
Definicja odpowiedniego interfejsu znajduje się w Załączniku VIII.

4.4.2 Graficzne środowisko edycji danych i prezentacji działania algorytmu

Jak już to zostało wcześniej powiedziane, przygotowywany algorytm ma być częścią większego systemu wspomagającego planowanie podróży. Niestety w czasie jego tworzenia nie były jeszcze dostępne inne moduły, gdyż powstawały one równolegle i były tworzone przez inne osoby. Ich brak musiał zostać uzupełniony tymczasowymi danymi i autorskimi strukturami danych. Aby umożliwić łatwe generowanie takich danych został w pierwszej kolejności stworzony graficzny edytor, który umożliwia rozmieszczanie na mapie miast oraz stacji, a także definiowanie rozkładów jazdy. Jego dodatkową cechą jest to, że umożliwia on również testowanie i wizualną prezentację etapów działania algorytmu planowania poprzez nanoszenie na mapę wygenerowanych podziałów podróży na etapy oraz ich późniejszych implementacji konkretnymi połączeniami – już ostatecznych wersji planów. Możliwe jest także sortowanie znalezionych rozwiązań wg różnych kryteriów, w tym wg ocen wystawionych przez moduł oceny propozycji podróży.

Podczas definiowania nowych rozkładów jazdy, użytkownik wskazuje kolejne stacje dla nowego połączenia, a system ustawia dla nich domyślne czasy w zależności od środka transportu (każdy ma inną prędkość poruszania się) – sprawia to, że tworzone rozkłady jazdy są bardziej naturalne. Definiowanie czasu połączeń odbywa się poprzez podanie czasu odjazdu z pierwszej stacji i czasów dla kolejnych stacji (te mogą być w postaci absolutnej lub względem odjazdu ze stacji pierwszej). Dodatkowo można określić dni tygodnia, w które dane połączenie obowiązuje. Aby ułatwić tworzenie większej ilości połączeń można także określić powtórzenia w ciągu jednego dnia – dwie godziny graniczne i co ile minut między nimi ma być odjazd (to jednak, zgodnie z definicją połączenia z rozdziału 4.4.3.10, oznacza stworzenie odpowiedniej ilości połączeń dla każdego powtórzenia wewnątrz jednego dnia). Można także od razu wygenerować połączenia powrotne – odchodzą one o tych samych porach ze stacji końcowej, co

oryginalne z początkowej; podróż na poszczególnych odcinkach trwa tyle samo. Przykład edycji rozkładu jazdy znajduje się poniżej (Rysunek 15).



Rysunek 15 Edycja rozkładu jazdy

4.4.3 Opis struktur danych

W tym rozdziale zostanę pokrótce przedstawione najważniejsze z wykorzystanych struktur danych wraz z opisami ich wykorzystania oraz dyskusją, dlaczego tak zostały opisane informacje powiązane z podróżowaniem.

4.4.3.1 Klasa Location

Opis

Opisuje położenie punktu.

Pola

- Współrzędne geograficzne punktu.
- Nazwa kontynentu, na którym dany punkt się znajduje.
- Nazwa wyspy, na której dany punkt się znajduje (możliwa jest specjalna wartość „część kontynentalna”).

Funkcjonalność

Klasa ta opisuje położenie geograficzne danego punktu. Oprócz samych współrzędnych zawiera ona także informację o nazwie kontynentu i wyspy, na których dany punkt się znajduje. Dane dodatkowe nie muszą być obecne. Jeśli jednak są dostępne, to istnieje możliwość korzystania z większej ilości reguł podczas procesu planowania podróży. Współrzędne geograficzne są dostępne zawsze.

4.4.3.2 Klasa WorldObject

Opis

Obiekt należący do świata.

Pola

- Lokacja danego obiektu w świecie.
- Nazwa danego obiektu.

Funkcjonalność

Jest to podstawowa klasa, z której dziedziczą wszystkie obiekty, które można umieścić w świecie. Obsługuje ona lokację danego obiektu oraz jego nazwę (ta nie musi być unikatowa w skali świata – może to być nazwa miejscowości lub np. dworca kolejowego).

4.4.3.3 Klasa MeanOfTransportation

Opis

Zawiera stałe identyfikujące możliwe środki transportu.

Pola

Stałe:

- AIRPLANE – samolot
- TRAIN – pociąg
- BUS – autobus
- SHIP – statek
- TAXI – taksówka

- WALK – spacer

Funkcjonalność

Środki transportu są rozróżniane poprzez posiadanie różnych, unikatowych identyfikatorów.

4.4.3.4 Klasa City

Opis

Opisuje miasto.

Pola

Nie zawiera żadnych pól oprócz odziedziczonych.

Funkcjonalność

Klasa ta dziedziczy z klasy WorldObject. Obiekty tego typu nie mają żadnego wpływu na algorytm planowania. W świecie umieszczane są tylko dla ułatwienia orientacji.

4.4.3.5 Klasa Station

Opis

Klasa bazowa opisująca stację.

Pola

- Środek transportu obsługiwany przez daną stację.

Funkcjonalność

Klasa bazowa dla wszystkich stacji. Dziedziczy z klasy WorldObject. Każda stacja ma przypisany do siebie obsługiwany przez nią środek transportu. W świecie realnym istnieją stacje, które obsługują więcej niż jeden środek transporty, lecz de facto jest to klika stacji, które leżą od siebie na tyle blisko, że mają tą samą nazwę.

4.4.3.6 Klasa Airport

Opis
Lotnisko.
Pola
Środek transportu = AIRPLANE.
Funkcjonalność
Dziedziczy z klasy Station.

4.4.3.7 Klasa BusStop

Opis
Przystanek autobusowy.
Pola
Środek transportu = BUS.
Funkcjonalność
Dziedziczy z klasy Station.

4.4.3.8 Klasa Harbour

Opis
Port.
Pola
Środek transportu = SHIP.
Funkcjonalność
Dziedziczy z klasy Station.

4.4.3.9 Klasa TrainStation

Opis
Stacja kolejowa.

Pola

Środek transportu = TRAIN.

Funkcjonalność

Dziedziczy z klasy Station.

4.4.3.10 Klasa Connection

Opis

Opisuje połączenie.

Pola

- Środek transportu.
- Lista odwiedzanych stacji.
- Lista czasów przyjazdu i odjazdu z każdej stacji.
- Flaga powtórzeń w poszczególne dni tygodnia.

Funkcjonalność

Klasa ta opisuje połączenia dla dowolnego środka transportu. Posiada ona uporządkowaną listę stacji, które są odwiedzane oraz odpowiadająca jej listę czasów przyjazdu i odjazdu z każdej z nich. Czasy są zapisane jako ilość minut od północy dnia przyjazdu do pierwszej ze stacji (czasu podstawienia). Oprócz tego specjalna flaga definiuje, w które dni tygodnia dane połączenie wyrusza. Jak łatwo zauważyć, połączenia o różnych godzinach (np. autobus o godzinie 14.00 i 14.20) są opisane przez dwie różne instancje tej klasy. Tak samo droga powrotna jest innym połączeniem. Taki zapis rozkładów jazdy ułatwia ich późniejsze wykorzystanie, a także eliminuje problem zjazdów do zajezdni, kiedy to trasa przejazdu jest inna niż zwykle.

Nie zostało to zaimplementowane, lecz w razie potrzeby można łatwo dodać dodatkowe zakresy rzeczywistych dat, kiedy to dane połączenie faktycznie obowiązuje. Na potrzeby testów nie było to jednak potrzebne.

4.4.3.11 Klasa World

Opis

Trzyma wszystkie informacje o obiektach świata i rozkładach jazdy.

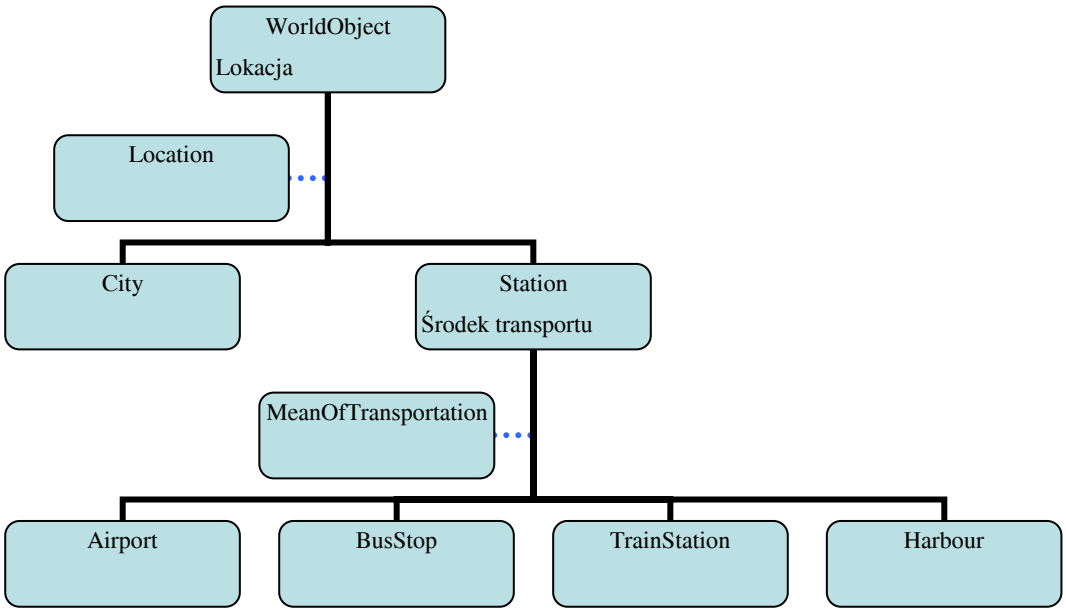
Pola

- Lista miast.
- Lista lotnisk.
- Lista przystanków autobusowych.
- Lista stacji kolejowych.
- Lista portów.
- Lista połączeń.

Funkcjonalność

Klasa ta jest wykorzystywana przez edytor do trzymania wszystkich informacji na temat obiektów istniejących w świecie oraz zdefiniowanych rozkładów jazdy.

4.4.3.12 Schemat dziedziczenia struktur danych



4.4.4 Planowanie – Etap I

W rozdziale tym zostaną opisane pewne wybrane szczegóły implementacyjne I etapu algorytmu planowania. Większość kwestii jest już jasna po wcześniejszym przedstawieniu całego algorytmu.

4.4.4.1 Wybór N stacji leżących najbliżej danego punktu

Jedną z bardzo często wykonywanych operacji podczas dzielenia podróży na etapy jest operacja wydobywania z bazy danych listy *n* stacji obsługujących dany środek transportu leżących najbliżej zadanego punktu. Tym punktem jest początek lub koniec odcinka, który należy podzielić w aktualnym kroku algorytmu dzielącego. Ponieważ operacja ta jest wywoływana wyjątkowo często, konieczne jest, aby działała maksymalnie szybko. Idealnie byłoby, gdyby sama baza danych była w stanie dokonać takiego doboru (wtedy wystarczające byłoby jednokrotne przejrzenie listy wszystkich stacji, możliwe, że nawet mniej, jeśli stosować optymalizacje dostępu do danych). W przygotowanej implementacji zostało jednak założone, że takiej możliwości nie ma. Oznacza to, że to algorytm planujący sam musi wybrać potrzebne *n* stacji. Ponieważ jednak proces planujący dysponuje ograniczoną ilością zasobów systemowych (zwłaszcza pamięci), to nie może sobie pozwolić na wyciągnięcie listy wszystkich stacji, nawet ograniczonej do dość dużego wycinka świata, gdyż byłby zmuszony do przetworzenia olbrzymiej ilości danych. Taka sytuacja nie może zostać zaakceptowana.

Aby rozwiązać ten problem, została zwiększona ilość zapytań do bazy danych. Każde kolejne zapytanie obejmuje coraz większy wycinek mapy świata. Pierwsze z nich obejmuje pewien kwadrat o środku w danym punkcie i o boku zależnym od środka transportu – im większa jest szansa na wystąpienie większej ilość stacji w okolicy danego punktu, tym bok kwadratu jest mniejszy. Oczywiste jest, że największe są szanse na dużą ilość przystanków autobusowych, mniejsze na stacje kolejowe, a najmniejsze na lotniska lub porty. Zapytanie dotyczy wszystkich stacji leżących w zadanym obszarze. Gdy zwrócona ilość stacji jest nie mniejsza od *n*, to wybierane jest z nich *n* najbliższych środka kwadratu. Gdy

jednak stacji jest zbyt mało, to bok kwadratu jest zwiększany dwukrotnie aż do skutku chyba, że zostanie osiągnięta długość maksymalna zdefiniowana jako półtora długości odcinka dzielonego na etapy (ogólnie: ε · długość odcinka dzielonego).

Wadą tego rozwiązania jest zwiększona ilość zadawanych zapytań do bazy danych, jednak ogranicza ono ilość jednocześnie przetwarzanych danych. Stanowi ono pewien kompromis pomiędzy obydwoma kryteriami oceny. Dobór początkowej długości boku kwadratu ma na celu zwiększenia szans na to, że wystarczy tylko jedno - dwa zapytania. Zakładając dodatkowo, że baza danych stosuje jednak pewne optymalizacje przy dostępie do danych, to możliwe, że każde zapytanie o ograniczonym zasięgu może być wykonane dość szybko i nie wymaga przejrzenia wszystkich znanych stacji na świecie.

4.4.4.2 Zastosowane reguły wyboru potencjalnych środków transportu między dwoma danymi punktami

Do wyboru potencjalnych środków transportu zostały zastosowane wymienione niżej reguły. Słowo „dystans" oznacza w nich odległość między danymi punktami. Reguły dotyczące kontynentów lub wysp są stosowane tylko, gdy te dane są dostępne dla obu punktów. A oto te reguły:

- Jeśli dystans jest większy lub równy minimalnej odległości dla samolotów, to dodaj do listy samolot. W przeciwnym wypadku dodaj samolot tylko, gdy oba punkty leżą na różnych kontynentach, lub na tym samym kontynencie, ale na różnych wyspach.
- Rozpatruj tylko, gdy dystans jest większy lub równy minimalnej odległości dla pociągów. Jeśli dane o kontynentach nie są dostępne, to dodaj pociąg do listy, a jeśli są dostępne, to dodaj go tylko, gdy oba punkty leżą na tym samym kontynencie i albo nie ma dostępnych danych o wyspie, lub te dane są dostępne i oba punkty leżą na tej samej wyspie.
- Rozpatruj tylko, gdy dystans jest większy lub równy minimalnej odległości dla autobusów. Jeśli dane o kontynentach nie są dostępne, to

dodaj autobus do listy, a jeśli są dostępne, to dodaj go tylko, gdy oba punkty leżą na tym samym kontynencie i albo nie ma dostępnych danych o wyspie, lub te dane są dostępne i oba punkty leżą na tej samej wyspie.

- Jeśli dystans jest nie mniejszy niż minimalna odległość dla statków, to dodaj rejs do listy.
- Jeśli dystans jest nie większy niż maksymalna odległość dla spaceru, to dodaj spacer do listy.
- Rozpatruj tylko, gdy dystans jest nie większy niż maksymalna odległość dla taksówki i nie mniejszy niż odległość minimalna oraz dodatkowo jest większy niż odległość maksymalna dla spaceru. Jeśli dane o kontynentach nie są dostępne, to dodaj taksówkę do listy, a jeśli są dostępne, to dodaj ją tylko, gdy oba punkty leżą na tym samym kontynencie i albo nie ma dostępnych danych o wyspie, lub te dane są dostępne i oba punkty leżą na tej samej wyspie.
- Na koniec usuń z listy wszystkie niedozwolone środki transportu przez użytkownika chyba, że oznaczałoby to usunięcie wszystkich możliwości.

Jak widać, zadaniem reguł jest wykluczenie możliwości, które na pewno nie mają sensu z punktu widzenia logiki – nikt nie będzie jechał autobusem przez morze! Wprowadzają one także pewną jakość do powstających planów, gdyż ograniczają, na przykład, odległość, jaką można pokonać pieszo lub taksówką. Wiadomo też, że nie nikt nie będzie leciał samolotem na zbyt krótkim dystansie. Zauważmy tutaj, że w przypadku tworzenia systemu profesjonalnego koniecznym będzie najprawdopodobniej rozszerzenie zbioru powyższych reguł, ale proponowane rozwiązanie jest dostatecznie elastyczne, aby nie stanowiło to problemu.

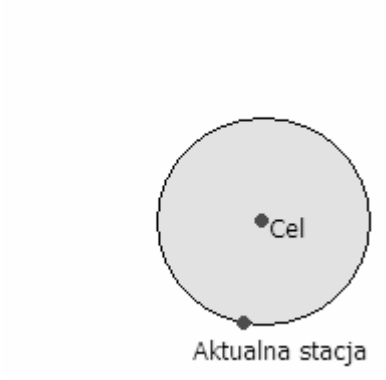
4.4.5 Planowanie – Etap II

W tym rozdziale zostaną opisane wybrane szczegóły implementacyjne II etapu algorytmu planowania. Podobnie jak poprzednio, większość kwestii powinna być już dosyć oczywista w kontekście omówionego wcześniej algorytmu.

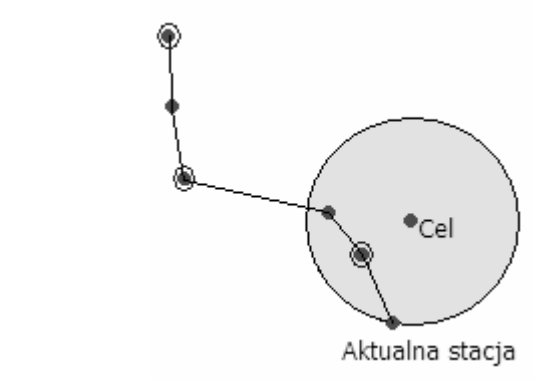
4.4.5.1 Reguły odrzucania potencjalnych przesiadek

Poniżej zostały przedstawione reguły wykorzystane do odrzucania potencjalnych przesiadek:

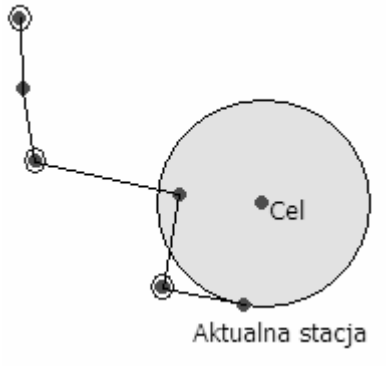
- Jeśli aktualne połączenie jest bezpośrednie, to sprawdź czy połączenie z danej przesiadki także jest bezpośrednie. Jeśli nie, to je odrzuć, a jeśli tak, to zaakceptuj je tylko wtedy, gdy dociera szybciej na stację końcową etapu niż połączenie aktualne.
- Porównaj następną stację w połączeniu w potencjalnej przesiadce z następną stacja w aktualnym połączeniu i pozwalaj na przesiadkę tylko wtedy, gdy jest to inna stacja lub można dojechać na tą samą stację szybciej.
- Jeśli odległości ze stacji następnej i ostatniej oraz ze stacji po środku między nimi (w sensie kolejności) w połączeniu przesiadkowym do stacji docelowej są większe niż odległość do tej stacji ze stacji aktualnej, to połączenie to można prawdopodobnie odrzucić. Aby się jednak upewnić, należy sprawdź jeszcze wszystkie pozostałe stacje w połączeniu przesiadkowym. Reguła ta odrzuca połączenia, które ewidentnie prowadzą w złą stronę. Wystarczy, aby przynajmniej jedna odległość z badanych była mniejsza niż odległość od stacji aktualnej, aby przesiadkę dopuścić - taka sytuacja oznacza, że połączenie będące kandydatem na przesiadkę w pewnym momencie zbliża się w dobrą stronę, czyli do końca etapu. Początkowe sprawdzenie tylko trzech stacji (próbkowanie) w niektórych przypadkach umożliwia uniknięcie konieczności sprawdzenia wszystkich stacji, co daje duże zyski, zwłaszcza, gdy tych stacji jest więcej. Poniższe rysunki ilustrują przeprowadzany test.
- Przy pomocy reguł można też odrzucać połączenia niespełniające ograniczeń narzuconych przez użytkownika.



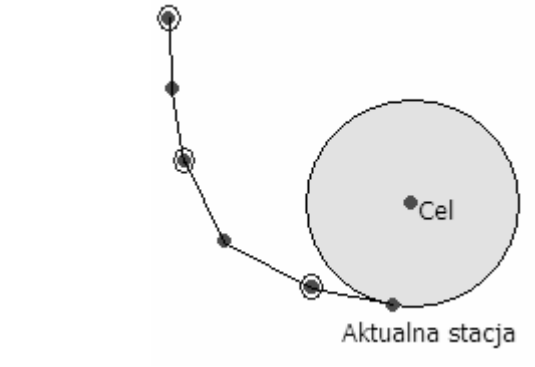
Rysunek 16 Aktualna stacja i cel etapu wraz z promieniem bliższości



Rysunek 17 Stacja próbkowa jest bliżej - warto się przesiąść



Rysunek 18 Dopiero sprawdzenie wszystkich stacji wykaże, że warto się przesiąść



Rysunek 19 Wszystkie kolejne stacje są dalej - odrzucenie przesiadki

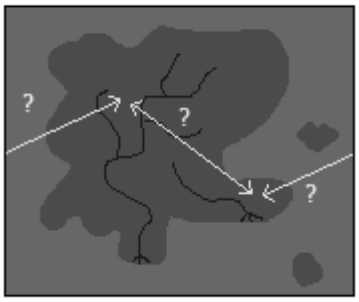
4.4.5.2 Przesiadki

Do opisu przesiadek mogą zostać wykorzystane zwykłe zadania używane do pamiętania aktualnego stanu wyborów podczas przeszukiwania rozkładów jazdy. Przesiadka może zostać zapamiętana poprzez ustawienie temu zadaniu danych roboczych zgodnie z połączeniem, na które ma się odbyć zmiana. Przez dane robocze rozumiana jest referencja do klasy połączenia, maska dnia tygodnia (definiuje, które z powtórzeń połączenia jest wykorzystywane) oraz indeks początkowej i aktualnej stacji w tym połączeniu. Dane te są potrzebne do dokonywanie przejść między kolejnymi stacjami podczas przeglądania możliwości podróży. Dokonanie przesiadki sprowadza się do skopiowania danych roboczych do aktualnego zadania. Dodatkowo należy wcześniej utworzyć nowy odcinek podróży na bazie starego zadania i dołączyć go do listy trzymanej razem

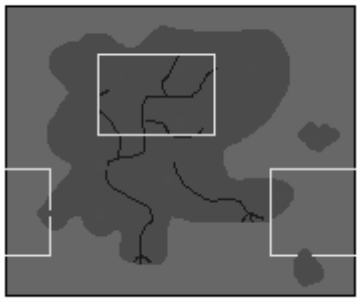
ze starym zadaniem. Tworzony odcinek opisuje pokonany dystans od czasu poprzedniej przesiadki (ewentualnie od początku podróży).

4.4.5.3 Okrągłość Ziemi

W przygotowanym edytorze (module geograficznym) świat jest prostokątną mapą. Aby jednak wprowadzić elementy realizmu, okrągłość Ziemi jest symulowana poprzez ciągłość mapy na styku wschodu i zachodu – świat rozciągnięty na ścianie walca. Z praktycznego punktu widzenia, wymusza to tylko branie tego faktu pod uwagę podczas liczenia odległości między dwoma punktami (branie minimum z odległości liczonej w obie strony - Rysunek 20) oraz podczas wyboru stacji leżących w pewnym prostokącie (może on przechodzić na drugi koniec mapy – Rysunek 21). Są to jedyne momenty, kiedy topologia świata ma znaczenie. Oznacza to również, że ewentualna implementacja pełnej kulistości Ziemi nie będzie problemem – będzie jedynie wymagała trochę bardziej skomplikowanych obliczeń (wszystkie wymagane modyfikacje odbywają się w obrębie modułu geograficznego).



Rysunek 20 Obliczanie odległości między dwoma punktami



Rysunek 21 Wybór stacji leżących w prostokątnym obszarze

4.4.6 Możliwości obliczeń równoległych

Pomimo stosowania mechanizmów ograniczających ilość przeglądanych możliwości w obu etapach algorytmu planującego podróż, ciągle istnieje ryzyko,

że będzie istniała konieczność przejrzenia ich znacznej ilości. Dlatego też warto zauważyć, iż zaproponowany algorytm umożliwia łatwe przeprowadzanie obliczeń równoległych, które umożliwi szybsze wykonanie całego procesu planowania.

W etapie pierwszym obliczenia można przeprowadzać równolegle podczas wyznaczania podziałów dla wykorzystanych stacji początkowych i końcowych (dokładniej: dla odcinków od początku danego etapu do stacji początkowej i od stacji końcowej do końca etapu). Natomiast w etapie drugim można równolegle rozwiązywać zadania odkładane w menadżerze zadań w ramach pojedynczego etapu, gdyż nie są one od siebie zależne lub równolegle implementować kilka podziałów na etapy. We wszystkich przypadkach występują również liczne odwołania do bazy danych, stąd dodatkowa możliwość pracy równoległej (nawet na jednej maszynie) - podczas oczekiwania na odpowiedź na zapytanie do bazy danych w jednym procesie można w innym wykonywać obliczenia związane już bezpośrednio z samym algorytmem planowania bez tracenia czasu na czekanie.

4.5 Personalizacja przy pomocy logiki przypadków

Jak już to zostało wcześniej stwierdzone w niniejszej pracy, ważnym elementem wspomagania podróżowania jest personalizacja. Na etapie planowania trasy podróży może to być ocenianie przygotowanych propozycji. Wiadomo już, że może ich być sporo i dużym ułatwieniem dla użytkownika byłoby ich wstępne posortowanie według jego osobistych preferencji. Problem jednak polega na tym, iż trudno jest powiedzieć, które informacje brać pod uwagę dokonując takiego porządkowania. Co więcej, przy sortowaniu potrzebna jest jedna wartość charakteryzująca całą propozycję, którą można następnie porównywać z innymi, aby móc ustalić odpowiedni porządek. Niestety plan podróży charakteryzuje wiele czynników. Między innymi są to: ogólna długość trasy, wykorzystane środki transportu, ilość przesiadek, czas oczekiwania na przesiadki, itd. Znaczenie może mieć również część świata, w której odbywa się podróż – użytkownik może woleć podróżować pociągiem po Szwajcarii (ze względu na piękne widoki) a samolotem nad pustyniami Australii (mniej różnorodnych widoków). Lokalizacja podróży

może mieć także bardziej lokalne znaczenie – na przykład, ktoś może lubić jeździć do pracy tramwajem, ale wracać już autobusem. Widać jasno, że kryteriów oceny może być wiele i nie ma żadnych przeciwwskazań, aby nie mogły one być ze sobą łączone. Co gorsza, każda podróż z reguły jest inna – zawiera inne dane charakteryzujące ją i/lub inną ich ilość niż inne podróże. Na szczęście, istnieje mechanizm umożliwiający rozwiązanie tego problemu poprzez kompleksowe spojrzenie na każdy przypadek i możliwość porównywania niejednorodnych opisów planów podróży. Tym rozwiązaniem jest logika przypadków.

4.5.1 Działanie logiki przypadków

Działanie logiki przypadków opiera się na założeniu, że podobne problemy będą posiadały podobne rozwiązania lub będą podjęte wobec nich podobne decyzje. Spostrzeżenie to jest oparte na obserwacji postępowania ludzi, gdzie często to doświadczenie jest najważniejszym czynnikiem decydującym o tym, jak kto sobie radzi z nowymi problemami; doświadczenie często jest ważniejsze od wiedzy. Prekursorzy logiki przypadków Riesback i Schank napisali, że „ludzcy eksperci nie są systemami reguł, ale bibliotekami doświadczenia”[12]. Ludzie rozwiązują wiele problemów szukając analogi do swoich poprzednich doświadczeń i próbują rozwiązywać nowe problemy poprzez przenoszenie starych rozwiązań na stojące przed nimi zadania. Ponieważ dwa problemy z reguły jednak różnią się od siebie, to i ich rozwiązania nie są identyczne. Trzeba wtedy zaadaptować posiadane rozwiązanie do nowego problemu poprzez wprowadzenie w nim odpowiednich zmian. Jeśli jest to możliwe, należy w takich sytuacjach wyodrębnić różnice w problemach i wprowadzić w rozwiązaniu jednego problemu zmiany odpowiadające tym różnicom, aby uzyskać odpowiednie rozwiązanie. Gdy jednak nie jest to możliwe, to zakłada się, że oba problemy będą posiadały takie same rozwiązania, czyli po prostu kopiuje się rozwiązanie jednego problemu do drugiego. Gdy już uda się rozwiązać nowy problem, to zapamiętuje się jego opis i odpowiadające mu rozwiązanie w bazie wiedzy, gdzie będzie on

[12] Riesback i Schank, 1989 r.

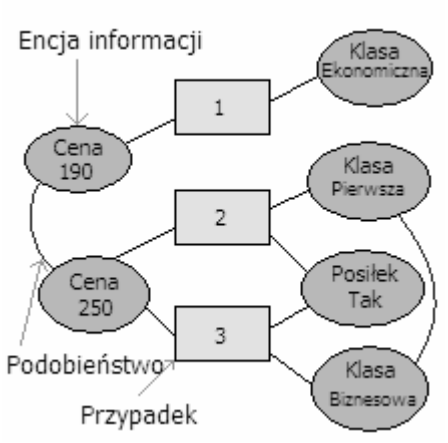
dostępny do ponownego wykorzystania w przyszłości. W ten sposób w systemie realizuje się pewną formę uczenia.

4.5.2 Sieci Wyciągające Przypadki

Aby jednak korzystać z logiki przypadków potrzebny jest mechanizm, który umożliwi przechowywanie starych przypadków i wyszukiwanie wśród nich podobnych do nowego problemu, który musi zostać rozwiązany. Mechanizmem takim są *Sieci Wydobywające Przypadki* (ang. *Case Retrieval Nets*). Zostaną one teraz pobieżnie opisane. Opis nie będzie bardzo dokładny z uwagi na fakt, iż mechanizm ich działania nie stanowi tematu niniejszej pracy. Dokładniejsze opisy mogą zostać znalezione w literaturze.

4.5.2.1 Opis przypadku

Każdy z przypadków zapisany w sieci jest opisany poprzez zbiór *Encji Informacji* (ang. *Information Entities*), który go definiuje. Każda encja to para: nazwa atrybutu – jego wartość. Do każdego przypadku może zostać przypisanych dowolnie wiele atrybutów (niekoniecznie o różnych nazwach). Encje są przypisywane do przypadków poprzez łuki przynależności. Aby zaoszczędzić pamięć i później ułatwić wyszukiwanie przypadków, zakłada się, że każda encja informacji może należeć do więcej niż jednego przypadku. Dodatkowo między encjami mogą występować łuki podobieństwa, które definiują jak bardzo obie encje są do siebie podobne. Ich brak oznacza zupełny brak podobieństwa. Zbiór wszystkich encji, przypadków oraz łuków tworzy sieć. Przykład takiej sieci jest widoczny poniżej (Rysunek 22). W praktyce nie stosuje się łuków podobieństwa, lecz zastępuje się je funkcjami, które obliczają podobieństwo dwóch encji opisujących ten sam atrybut. Dla różnych atrybutów przyjmuje się podobieństwo 0. Można jednak czynić wyjątki od tej reguły.



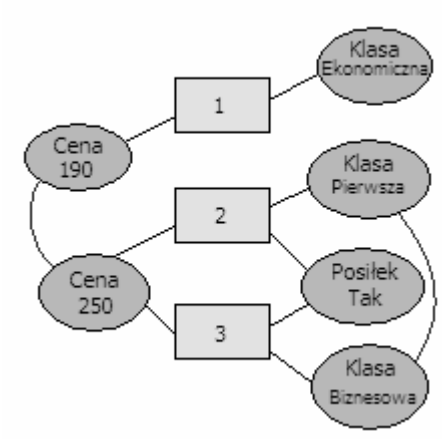
Rysunek 22 Przykład *Sieci Wyciągającej Przypadki* z zapisanymi trzema przypadkami

4.5.2.2 Wydobywanie przypadków

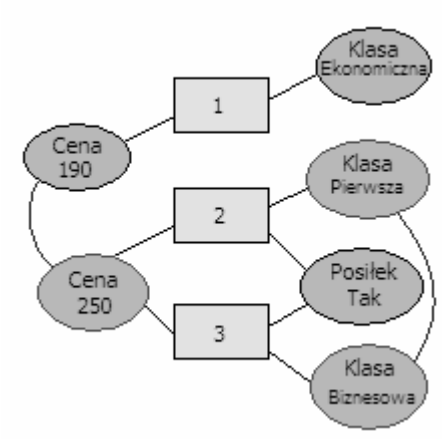
Wydobywanie przypadków jest najczęściej wykonywaną operacją przez sieć ze względu na jej zadania. Musi ona wobec tego być wykonywana maksymalnie szybko. Jest to możliwe dzięki opisanej wcześniej architekturze sieci. Największą zaletą *Sieci Wydobywających Przypadki* jest to, że nie muszą one przejrzeć wszystkich zapisanych w nich przypadków, aby znaleźć te najbardziej podobne do zadanego. Proces wydobywania podobnych przypadków przebiega trzyetapowo:

1. Najpierw aktywowane są wszystkie *Encje Informacji* podobne do tych z opisu problemu do rozwiązania.
2. Następnie aktywacje tych encji są propagowane na przypadki, do których one należą.
3. Na koniec zbierane są najbardziej podobne przypadki z tych aktywowanych.

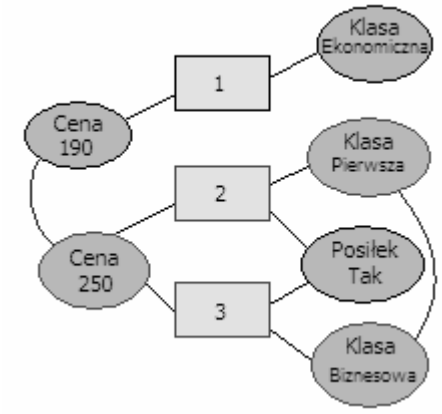
W ten sposób nie ma konieczności zaglądania do każdego przypadku i sprawdzania jak bardzo jest podobny do tego, dla którego szukamy rozwiązania. Poniżej został przedstawiony przykład wyszukiwania najbardziej podobnego przypadku do takiego, w którym cena wynosi 270, a klasa jest biznesowa:



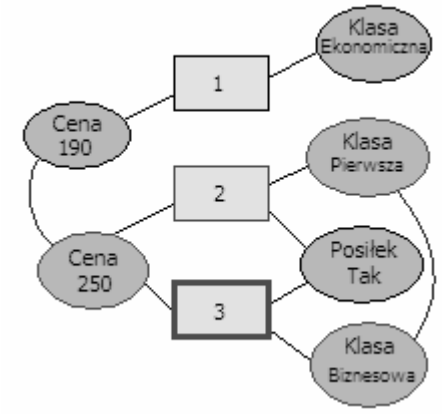
Rysunek 23 Sieć w spoczynku



Rysunek 24 Aktywacji encji informacji



Rysunek 25 Propagacja aktywacji na przypadki



Rysunek 26 Zebranie aktywnych przypadków i wybór najbardziej podobnego z nich

4.5.2.3 Porównywanie przypadków

Porównywanie przypadków polega na obliczeniu ważonej sumy podobieństw wszystkich *Encji Informacji,* przy czym podobieństwo jest liczone między wszystkimi encjami o danej nazwie atrybutu w obu porównywanych przypadkach. Oznacza to, iż waga jest aplikowana podobieństwu obliczonemu całej grupie *Encji Informacji* o danej nazwie atrybutu, a nie każdemu z osobna. Takie podejście umożliwia radzenie sobie z przypadkami, gdy porównywane przypadki posiadają różną ilość *Encji Informacji* o danej nazwie atrybutu (w szczególności, gdy jeden z nich ich nie ma). Szczegóły porównywania całych grup nie są istotne z punku widzenia tej pracy. Istotne są tylko dwa fakty: to, że

przy pomocy wag można definiować istotność atrybutów i to, że można wyznaczyć liczbę z zakresu [0..1] opisującą podobieństwo dwóch różnych przypadków. Warto także zauważyć, że do porównywania różnych atrybutów można stosować różne funkcje porównujące, co daje dodatkowe możliwości konfiguracji zachowania się sieci.

4.5.3 Zastosowanie logiki przypadków do personalizacji planowania podróży

Logikę przypadków - a bardziej konkretnie - *Sieci Wyciągające Przypadki* można wykorzystać do porównywania propozycji między sobą [26, 31]. Przy ich pomocy można wyznaczać podobieństwo danego przypadku do danych historycznych. Jeśli dodatkowo dla tych informacji z przeszłości będzie się pamiętać, czy dana propozycja została przez użytkownika zaakceptowana, czy też zdecydowanie wskazana jako nieodpowiednia[13], to będzie wiadomo, jak najbardziej podobny przypadek z przeszłości został potraktowany przez użytkownika. Jasne jest, że najlepszymi z nowych propozycji będą te, które będą najbardziej podobne do propozycji zaakceptowanych kiedyś przez użytkownika, a najgorszymi te, które będą najbardziej podobne do propozycji odrzuconych. Jako jakość propozycji możemy przyjąć uzyskane podobieństwo do przypadku historycznego brane z minusem, gdy była to propozycja odrzucona. Stąd każdej nowej propozycji będziemy mogli przypisać wartość z przedziału [-1;1]. Najlepsze będą te z 1, a najgorsze te z -1.

Po wybraniu przez użytkownika pewnej propozycji (lub jej odrzuceniu), do danych historycznych można dodać nowy opis podróży wraz z decyzją podjętą wobec niego. W ten sposób baza przypadków historycznych będzie rosła i dzięki temu cały mechanizm będzie działał lepiej, gdyż będzie coraz więcej przypadków, do których te nowe będą porównywane. Zapamiętywanie nowych przypadków realizuje proces uczenia się całego mechanizmu oceniającego.

[13] Nie chodzi tu o fakt jej nie wybrania, ale o jawne stwierdzenie przez użytkownika, że ta propozycja mu zupełnie nie odpowiada.

Opisany mechanizm można także wykorzystać do innych celów. Można gromadzić nie tylko dane historyczne dotyczące samych podróży, ale także zapytań wydawanych przez użytkownika. Takie informacje mogłyby być wykorzystywane do uzupełniania niepełnych zapytań.

4.5.4 Opis historii podróży

Przypadek podróży jest opisywany przez następujący zbiór atrybutów:

Atrybut	Waga	Opis
Wykorzystane środki transportu	Ważne (3,0)	Dla każdego wykorzystanego środka transportu dodawana jest jedna encja informacji „Wykorzystany środek transportu". Sprawia to, że będą do siebie podobne te przypadki podróży, które były pokonywane tymi samymi środkami transportu.
Godzina odjazdu	Nieważne (0,5)	Do opisu dodawana jest także encja informacji zawierająca godzinę rozpoczęcia podróży (bez minut). Celem tego atrybutu jest wprowadzenie podobieństwa podróży odbywających się o tej samej porze dnia. Jest to jednak kryterium drugorzędne i powinno mieć znaczenie, tylko wtedy, gdy inne atrybuty są takie same.
Ranking środków transportu	Neutralne (1,0)	Każdemu wykorzystanemu środkowi transportu przypisana jest pozycja w rankingu ilości odcinków pokonanych przy jego pomocy. Umożliwia to porównywanie głównych środków transportu. Stąd

		podróże, które są pokonywane głównie tymi samymi środkami transportu będą do siebie bardziej podobne.
Lokacja początkowa	Neutralne (1,0)	Umożliwia porównywanie początków podróży. Lokacje początkowa i końcowa są mniej ważne niż pierwsza i ostatnia stacja, gdyż przynajmniej jedna z nich może być często taka sama (np. dom). Stacje mają intuicyjnie więcej wspólnego z trasą, jaka jest pokonywana.
Lokacja końcowa	Neutralne (1,0)	Umożliwia porównywania końców podróży.
Pierwsza stacja	Ważne (3,0)	Pierwsza z odwiedzonych stacji, o ile istnieje. Początek i koniec podróży są ważne, gdyż użytkownik może mieć przyzwyczajenia, do podróżowania na pewnych trasach w określony sposób.
Ostatnia stacja	Ważne (3,0)	Ostatnia z odwiedzonych stacji, o ile istnieje.
Całkowity czas podróży.	Neutralne (1,0)	Zapamiętywanie całkowitego czasu podróży wyrażonego w minutach zapewnia większe podobieństwo podróży, które trwają mniej więcej tyle samo.
Ilość przesiadek	Neutralne (1,0)	Ilość przesiadek umożliwia porównywanie etapowości (stopnia skomplikowania) poszczególnych podróży.

Istnieją także inne atrybuty podróży, które zostały pominięte, gdyż nie były zaimplementowane. Są to: cena podróży, średnia cena za odcinek podróży, klasy przedziałów, czy był to przedział dla niepalących, serwowane posiłki, to czy miejsce było przy oknie, itd.

4.6 Przedstawienie dodatkowych zagadnień

W rozdziale tym zostaną przedstawione pomysły, które nie zostały zaimplementowane, a które mogą usprawnić działanie algorytmu planującego. Przedstawione zostaną również możliwości pełnego wykorzystania algorytmu.

4.6.1 Unikanie rejsów dookoła lądu

Statki są dość specyficznym środkiem transportu, ponieważ poruszają się po wodzie, a punkty, pomiędzy którymi kursują, czyli porty, znajdują się na lądzie. W związku z tym, czasami dochodzi do sytuacji, że muszą one pokonywać bardzo duże dystanse, mimo iż porty nie są od siebie znacznie oddalone. Dla przykładu załóżmy, że podróż ma się odbyć z Puli w Chorwacji do Marsylii we Francji. Pokonując ten odcinek statkiem, musimy opłynąć cały półwysep Apeniński. Korzystniej jednak by było dopłynąć do wschodniego wybrzeża Włoch, przesiąść się, na przykład, w pociąg, dotrzeć na brzeg zachodni, gdzie znowu można by się przesiąść w statek i spokojnie dopłynąć do Marsylii (Rysunek 27). Takie rozwiązanie byłoby zdecydowanie szybsze. Problemem jest jednak uwzględnianie takich niuansów podczas planowania. Aby uniknąć opływania lądu, należałoby dodać kilka dodatkowych etapów podróży do przygotowywanego podziału w etapie I (tak jak w przedstawionym przykładzie). To jednak wymaga sprawdzenia, czy w „okolicy" linii łączącej dwa porty nie znajdują się jakieś inne leżące na innych lądach. Takie sprawdzenie byłoby jednak procesem długotrwałym, gdyż – abstrahując już od definicji „w pobliżu" - trzeba by sprawdzić bardzo dużą ilość portów i dodatkowo uporządkować je tak, aby ustalić nowe etapy podróży... Dodatkowo byłby problem ze stwierdzeniem, które z tych portów w ogóle warto brać pod uwagę, które z nich rzeczywiście znajdują się na opływanym lądzie, a które na przykład na brzegu, wzdłuż którego się płynie przez pewien czas i który nie ma nic wspólnego z opływanym lądem. Nawet w przytoczonym przykładzie wszystkie wymienione porty leżą na kontynentalnej części Europy. Branie ich wszystkich pod uwagę sprawiłoby, że w etapie II byłoby niezmiernie więcej możliwości do przejrzenia, co oznaczałoby znaczący wzrost czasu wykonania, a to byłoby już nie do zaakceptowania.

Rysunek 27 Konieczność opływania lądu[14]

Pewnym rozwiązaniem tego problemu mogą być reguły eksperckie. Jak już było to wcześniej wspomniane, mogą one wymuszać rozpatrzenie pewnych etapów na algorytmie planującym. Nic nie stoi na przeszkodzie, aby wykorzystać je do dodania różnych odcinków do planowanej trasy podróży. W przykładzie dotyczącym podróży z Puli do Marsylii mogłyby one dodać różne odcinki kolejowe łączące oba brzegi Włoch, a potem algorytm już sam w naturalny sposób rozwiązałby problem dotarcia do odpowiednich portów. Oznacza to jednak, że takich reguł byłoby potrzebnych bardzo dużo. Oczywiście rodzi się pytanie, kto miałby je tworzyć i czy miałyby one być zakodowane w algorytmie na stałe, czy być wczytywane z plików konfiguracyjnych (w pierwszym przypadku działałyby szybciej)? Musiałby także zostać stworzony system, który szybko decydowałby, które z nich mogą mieć zastosowanie, aby nie było konieczności przeglądania ich wszystkich. To akurat jest możliwe, gdyż można przyjąć, że reguły mogą mieć zastosowanie tylko do pewnego prostokątnego obszaru na mapie, a z tego można już zbudować szybkie testy działające dla

[14] Oryginalna mapa pochodzi z Google Maps [37].

całych grup reguł. Można także zauważyć, że tak naprawdę opływanie lądów może mieć znaczenie tylko przy stosunkowo krótkich rejsach – takich jak właśnie po Morzu Śródziemnym. Przy rejsach długodystansowych, takie opłynięcia są pomijalne.

4.6.2 Planowanie podróży między wieloma punktami

Przedstawiony algorytm planowania umożliwia przygotowanie planu podróży pomiędzy dwom dowolnymi punktami przy zadanych ograniczeniach czasowych. Warto się jednak także zastanowić nad możliwością zaplanowania podróży pomiędzy większą ilością takich punktów. Tymi punktami mogą być, na przykład, obiekty uznane przez moduł personalizacyjny za ciekawe i warte zobaczenia podczas zwiedzania jakiegoś miasta lub regionu, albo jakieś miejsca podane przez użytkownika, które chce on odwiedzić po drodze. W pierwszym przypadku pojawia się problem ustalenia kolejności odwiedzania tych miejsc (problem komiwojażera). W drugim przypadku takiego problemu już nie ma, gdyż to użytkownik podaje kolejność.

Zakładając jednak, że kolejność jest znana, można wykorzystać przedstawiony algorytm do zaplanowania podróży pomiędzy poszczególnymi parami punktów. Planowanie można wtedy wykonywać para za parą (od początku do końca lub odwrotnie – wybór kolejności określa stosowane limity czasowe; poprzednia para ogranicza ramy czasowe kolejnej). Takie podejście nie gwarantuje jednak znalezienia optymalnego planu dla wszystkich punktów. Aby je znaleźć należałoby rozwiązywać pary w każdej możliwej kolejności, dodatkowo pamiętając o tym, że dla każdej pary możemy uzyskać wiele rozwiązań, z których każde należałoby rozpatrzyć oddzielnie. Jest to wykonalne w rozsądnym czasie tylko dla bardzo małej ilości punktów.

4.6.3 Problemy niewiedzy

Algorytm planujący musi stawić czoło pewnym problemom niewiedzy. Nie jest on w stanie z absolutną pewnością przewidzieć czasu, na przykład, jazdy taksówką, gdyż ten zależy od bardzo wielu czynników. Konsekwencją tego jest niepewność przewidzenia opłaty za przejazd. Algorytm też nie ma w tej chwili możliwości sprawdzania czy między dwoma punktami, które leżą kilkadziesiąt metrów od siebie, nie ma ruchliwej drogi, albo rzeki, co może znacząco wydłużyć czas pokonania dystansu między nimi – może się nawet okazać, że spacer, który wydawał się najbardziej odpowiedni, nie wchodzi już w grę. Środkiem przeciwdziałającym estymacji czasu jest jego rezerwowanie (rozdział 4.2.2.5) – przyjmowana jest pewna ilość czasu, która powinna wystarczyć na pokonanie danego odcinka podróży. W skrajnych przypadkach może się ona jednak okazać niewystarczająca. Często będzie ona też zbyt długa, co wydłuży faktyczny czas oczekiwania na przesiadkę. Obie sytuacje mogą jednak być poprawione ręcznie przez użytkownika (rozdział 4.6.5). Cenę za przejazd także można ograniczyć z góry pewną rozsądną wartością – podczas odbywania podróży użytkownik zapłaci najwyżej mniej niż się spodziewał. Do uwzględniania przeszkód terenowych potrzebna są zaś dane geodezyjne – za korzystanie z nich trzeba jednak płacić. Zakładając jednak, że są one dostępne, to można przy ich pomocy stworzyć dodatkowe reguły wybierające potencjalne środki transportu do pokonania określonego odcinka podróży.

4.6.4 Planowanie bez wiedzy o pozycjach stacji

Można także rozważyć przypadek planowania, gdy nie ma się do dyspozycji pozycji stacji (współrzędnych geograficznych). W takiej sytuacji trudno jest łączyć ze sobą różne środki transportu, gdyż nie ma możliwości stwierdzenia, które stacje innych środków transportu są blisko stacji, którą aktualnie rozpatrujemy. W efekcie, trudno jest zbudować podział podróży na etapy. Potrzebne są jawne wskazania mówiące, że z tej stacji jest blisko do innej, lub można do niej dotrzeć przy pomocy określonego środka transportu. Z reguły, gdy nie dysponuje się wiedzą o współrzędnych, możliwe jest planowanie w

obrębie tylko jednego środka transportu – jednak i wtedy jest do rozpatrzenia więcej możliwości ze względu na mniejsze możliwości zastosowania reguł odrzucających część możliwości. Jest to de facto planowanie ze zdegenerowanym, jednoetapowym podziałem podróży wygenerowanym w etapie I (jednak obie stacje trzeba jawnie wskazać, na przykład, poprzez podanie ich nazw – jednak mało w tym automatyzmu, a systemy realizujące takie zadania są dostępne już dziś). Jedynymi informacjami, na podstawie których można by wywnioskować pewne fakty, są informacje o kontynencie, na którym dana stacja się znajduje, lub kraju, w którym ona leży. Dane te można niejako przypisać domyślnie podczas dodawania do systemu stacji określonego przewoźnika. Na przykład, wiadomo, że stacje PKP są w Europie, w Polsce. Nie można jednak wywnioskować z takich informacji, czy dane dwie stacje nie leżą, powiedzmy, na dwóch różnych wyspach. Oczywiście, podczas wyszukiwania połączenia w etapie II wyjdzie, że takiego połączenia nie ma; zajmie to jednak sporo cennego czasu. Konkludując, w przypadku planowania bez danych o położeniu stacji potrzebne są dodatkowe informacje łączące różne środki transportu, których zadaniem jest umożliwienie wygenerowania podziału podróży na etapy oraz można przypuszczać, że będzie istniała konieczność przeglądania większej ilości możliwości, ze względu na ograniczone możliwości stosowania reguł ograniczających przestrzeń poszukiwań.

4.6.5 Zwiększenie interakcji użytkownika z algorytmem planującym

Aby zwiększyć atrakcyjność planów dla użytkownika, można dać mu większy wpływ na kształt przygotowywanych planów. Można to osiągnąć poprzez danie mu możliwości edycji czy to przygotowywanych podziałów w etapie I, czy też już gotowych rozwiązań z etapu II.

Po etapie I użytkownik mógłby wybrać sobie jedną z przygotowanych możliwości i ją dowolnie modyfikować, aby bardziej odpowiadała jego celom. Mógłby zmienić trasę w taki sposób, aby przejechać przez jego ulubione miasto. Mógłby też zażądać, aby postój w tym mieście trwał dłużej, aby mieć czas na spacer lub krótkie spotkanie ze znajomymi. Powodów, dla których użytkownik

mógłby chcieć wprowadzać zmiany jest oczywiście o wiele więcej. Ważny jest jednak sam fakt, że takie rozwiązania znacznie zwiększyłoby jakość końcowego efektu w oczach użytkownika. Jego implementacja wiązałaby się z koniecznością stworzenia odpowiedniego interfejsu do dokonywania takich zmian w maksymalnie wizualny i intuicyjny sposób. Samo uwzględnienie w algorytmie modyfikacji nie byłoby problemem, gdyż albo byłyby to modyfikacje podziałów podróży na etapy, które nie mają znaczenia dla etapu II planowania, albo też byłby to rozszerzony zbiór ograniczeń, który miałby zastosowania jedynie w pewnych momentach – przesiadki na niektórych stacjach byłyby traktowane w sposób specjalny.

Modyfikacje wyników etapu II mogłyby zaś polegać na wyborze innego połączenia na każdym odcinku podróży. Użytkownik mógłby narzucić wybór wcześniejszego połączenia, jeżeli uzna, że zostało przewidziane za mało czasu na dojazd taksówką ze stacji kolejowej w centrum miasta na lotnisko leżące na jego obrzeżach. Mógłby też zdecydować się na późniejszy lot. Dokonanie takich zmian mogłoby jednak oznaczać konieczność przeplanowania znaczących części podróży.

4.7 Testy działania algorytmu

Ponieważ autorowi niniejszej pracy nie są znane inne systemu umożliwiające planowanie wykorzystujące różne środki transportu, nie jest możliwe przeprowadzenie testów porównawczych z takimi systemami. W takiej sytuacji testy działania algorytmu planowania podróży mają na celu zweryfikowanie, czy wykorzystywane reguły rzeczywiście przynoszą spodziewane rezultaty, to jest ograniczają ilość przeglądanych możliwości, a jednocześnie nie powodują nie znalezienia najlepszych z możliwych rozwiązań.

4.7.1 Sposób przeprowadzenia testów

Przeprowadzane testy polegały na zliczaniu zapytań do baz danych o listy stacji określonego środka transportu w podanym obszarze oraz zapytań o listy połączeń przechodzących przez podaną stację. Są to kluczowe elementy działania algorytmu planującego, gdyż oznaczają one zapytania do zewnętrznego źródła danych, co może być ważnym czynnikiem oceny skuteczności, na przykład, ze względu na czas ich realizacji. Obserwowane było działanie algorytmu w dla sześciu par lokacji:

- dwóch leżących blisko siebie (pary numer 1 oraz 2)
- dwóch leżących w średniej odległości (pary numer 3 oraz 4)
- dwóch leżących daleko od siebie (pary numer 5 oraz 6)

Dla każdej pary algorytm był uruchomiony dla następujących kombinacji ustawień opcji algorytmu:

	Użyj reguł doboru środków transportu	**Użyj reguł odrzucania przesiadek**	**Ograniczaj ilość połączeń startowych oraz alternatywnych przesiadek**	**Wykorzystuj dodatkowe dane geograficzne**
1	**X**[15]	**X**	**X**	**X**
2				
3	**X**			
4		**X**		
5			**X**	
6				**X**

Tabela 1 Użyte zestawy opcji podczas testów

[15] X oznacza włączenie opcji, puste pole oznacza jej wyłączenie

Oznacza to, że pomiary były dokonywane w 36 przypadkach. Ponieważ data, od której zaczyna się planowanie ma znaczenie dla przeglądanych możliwości (o różnych porach różnych dni tygodnia mogą być dostępne różne połączenia), to planowanie pomiędzy poszczególnymi parami lokacji odbywało się według następujących dat i godzin:

Numer pary lokacji	Data rozpoczęcia planowania
1	2005-09-10 godzina 10:50
2	2005-09-10 godzina 10:58
3	2005-09-10 godzina 11:08
4	2005-09-10 godzina 11:14
5	2005-09-10 godzina 11:18
6	2005-09-10 godzina 11:23

Tabela 2 Godziny oraz daty rozpoczęcia planowania dla poszczególnych par lokacji

Pozostałe parametry konfiguracji opcji takie same dla wszystkich testów:

Rodzaj planowania:	Planowanie od podanej daty
Czas bycia przed odlotem na lotnisku:	90 minut
Czas bycia przed rejsem w porcie:	45 minut
Dopuszczalne środki transportu:	Spacer, taksówka, autobus, samolot, statek, pociąg
Maksymalna ilość przesiadek:	10
Minimalny trwania przesiadki:	5 minut
Maksymalny czas trwania przesiadki:	120 minut

Tabela 3 Ustawienia pozostałych opcji podczas planowania

4.7.2 Dane testowe

Danymi testowymi były rozkłady jazdy zdefiniowane w przygotowanym edytorze. Należy zauważyć, iż rozkłady te nie są ze sobą „zsynchronizowane" w tym sensie, że były one układane w sposób losowy i niekoniecznie muszą istnieć

możliwości szybkich przesiadek pomiędzy poszczególnymi połączeniami, jak ma to miejsce w świecie rzeczywistym. Brak danych zsynchronizowanych wiąże się z faktem, iż ich generowanie byłoby zbyt pracochłonne. Nie oznacza to jednak, iż na podstawie tych danych nie można ocenić działania algorytmu. Brak synchronizacji połączeń będzie się objawiał jedynie dłuższymi przesiadkami.

4.7.3 Wyniki testów

Poniżej zostały przedstawione wyniki testów z rozbiciem na poszczególne pary lokacji (szczegółowe dane można znaleźć w arkuszu kalkulacyjnym stanowiącym załącznik do niniejszej pracy – Załącznik XI):

4.7.3.1 Para lokacji numer 1

Zestaw opcji	Ilość zapytań o stacje	Ilość zapytań o połączenia dla stacji	Przyrost zapytań o stacje	Przyrost zapytań o połączenia	Procent pogorszenia	
					Stacje	Połączenia
1	49466	521				
2	62386	576	12920	55	0,261	0,106
3	57993	576	8527	55	0,172	0,106
4	62386	521	12920	0	0,261	0
5	62386	576	12920	55	0,261	0,106
6	62386	576	12920	55	0,261	0,106

Tabela 4 Wyniki testów dla pary lokacji numer 1

Niezależnie od wykorzystanych opcji najkrótsze i najszybsze plany były takie same. Na podstawie danych statystycznych widać, że najlepsze rezultaty otrzymuje się wykorzystując wszystkie opcje. Widać znaczący wpływ wykorzystania reguł na ilość zapytań. Reguły doboru środków transportu działają lepiej, gdy mają do dyspozycji dodatkowe dane geograficzne.

4.7.3.2 Para lokacji numer 2

Zestaw opcji	Ilość zapytań o stacje	Ilość zapytań o połączenia dla stacji	Przyrost zapytań o stacje	Przyrost zapytań o połączenia	Procent pogorszenia	
					Stacje	Połączenia
1	12063	26				
2	14407	26	2344	0	0,194	0
3	14407	26	2344	0	0,194	0
4	14407	26	2344	0	0,194	0
5	14407	26	2344	0	0,194	0
6	14407	26	2344	0	0,194	0

Tabela 5 Wyniki testów dla pary lokacji numer 2

Niezależnie od wykorzystanych opcji zostały znalezione takie same plany. Na podstawie danych statystycznych widać, że najlepsze rezultaty otrzymuje się wykorzystując wszystkie opcje. Reguły dobory środków transportu nie dają zysku bez dodatkowych danych geograficznych. Natomiast reguły odrzucania przesiadek nic nie dały - jest to spowodowane niewielką odległością obydwu lokacji.

4.7.3.3 Para lokacji numer 3

Zestaw opcji	Ilość zapytań o stacje	Ilość zapytań o połączenia dla stacji	Przyrost zapytań o stacje	Przyrost zapytań o połączenia	Procent pogorszenia	
					Stacje	Połączenia
1	84398	249				
2	97498	291	13100	42	0,155	0,169
3	92498	291	8100	42	0,096	0,169
4	97498	255	13100	6	0,155	0,024
5	97498	285	13100	36	0,155	0,145
6	97498	291	13100	42	0,155	0,169

Tabela 6 Wyniki testów dla pary lokacji numer 3

Niezależnie od wykorzystanych opcji zostały znalezione takie same plany. Ponownie najlepsze rezultaty otrzymuje się wykorzystując wszystkie opcje. Widać, że reguły doboru środków transportu działają lepiej, gdy mają do

dyspozycji dodatkowe dane geograficzne. Widać także działanie reguł odrzucających przesiadki. Przyrost ilości zapytań o połączenia w zestawie 4 jest spowodowany większą ilością rozpatrywanych etapów, gdyż nie były wykorzystane reguły doboru środków transportu.

4.7.3.4 Para lokacji numer 4

Zestaw opcji	Ilość zapytań o stacje	Ilość zapytań o połączenia dla stacji	Przyrost zapytań o stacje	Przyrost zapytań o połączenia	Procent pogorszenia	
					Stacje	Połączenia
1	42903	1353				
2	60305	2607	17402	1254	0,406	0,927
3	53765	2607	10862	1254	0,253	0,927
4	60305	2245	17402	892	0,406	0,659
5	60305	2607	17402	1254	0,406	0,927
6	60305	2607	17402	1254	0,406	0,927

Tabela 7 Wyniki testów dla pary lokacji numer 4

Niezależnie od wykorzystanych opcji najkrótsze i najszybsze plany były takie same. Najlepsze wyniki uzyskuje się korzystając ze wszystkich opcji. Reguły doboru środków transportu po raz kolejny działają lepiej, gdy są dostępne dodatkowe dane geograficzne. Reguły odrzucające przesiadki umożliwiają ograniczenie ilości zapytań o połączenia prawie o połowę. Wzrost zapytań o połączenia w zestawie 4 znów można tłumaczyć większą ilością podziałów na etapy spowodowaną wyłączeniem reguł doboru środków transportu.

4.7.3.5 Para lokacji numer 5

Zestaw opcji	Ilość zapytań o stacje	Ilość zapytań o połączenia dla stacji	Przyrost zapytań o stacje	Przyrost zapytań o połączenia	Procent pogorszenia	
					Stacje	Połączenia
1	86253	291				
2	111987	509	25734	218	0,298	0,749
3	106771	509	20518	218	0,238	0,749
4	111987	412	25734	121	0,298	0,416
5	111987	509	25734	218	0,298	0,749
6	111987	509	25734	218	0,298	0,749

Tabela 8 Wyniki testów dla pary lokacji numer 5

Niezależnie od wykorzystanych opcji zostały znalezione takie same plany. I tym razem najlepsze rezultaty uzyskuje się wykorzystując wszystkie opcje. Reguły doboru środków transportu działają znacząco lepiej, gdy są dostępne dodatkowe dane geograficzne. Reguły odrzucające przesiadki pozwalają na redukcję ilości zapytań o połączenia o prawie 60%.

4.7.3.6 Para lokacji numer 6

Zestaw opcji	Ilość zapytań o stacje	Ilość zapytań o połączenia dla stacji	Przyrost zapytań o stacje	Przyrost zapytań o połączenia	Procent pogorszenia	
					Stacje	Połączenia
1	45128	74				
2	58261	288	13133	214	0,291	2,892
3	55397	288	10269	214	0,228	2,892
4	58261	246	13133	172	0,291	2,324
5	58261	288	13133	214	0,291	2,892
6	58261	288	13133	214	0,291	2,892

Tabela 9 Wyniki testów dla pary lokacji numer 6

Niezależnie od wykorzystanych opcji zostały znalezione takie same plany. Również i w tym przypadku najlepsze rezultaty uzyskuje się wykorzystując

wszystkie opcje. Reguły doboru środków transportu działają zdecydowanie lepiej, gdy mają do dyspozycji dodatkowe dane geograficzne. Reguły odrzucające przesiadki pozwalają na pewne ograniczenie ilości zapytań o połączenia. W połączeniu z regułami doboru środków transportu jest już to prawie 70% redukcji.

4.7.4 Wnioski z testów

Z przedstawionych zestawień wyników widać jasno, że we wszystkich przypadkach najmniej odwołań do obu baz danych miało miejsce w momencie, gdy były włączone wszystkie opcje ograniczające ilość rozpatrywanych możliwości (zestaw 1). Co więcej, we wszystkich przypadkach mimo przejrzenia mniejszej ilości możliwości, najlepsze rozwiązania były takie same jak najlepsze ze znalezionych przy użyciu pozostałych zestawów opcji. Oznacza to, że użycie opcji przyspieszających wyszukiwanie nie powoduje nie znalezienia wartościowych rozwiązań. Warto także odnotować, że w 4 na 6 przypadków algorytm znajdował te same rozwiązania bez względu na wykorzystywany zestaw opcji.

Zyski z wykorzystania opcji przyspieszających wyszukiwanie były na tyle duże, że ilości zapytań o stacje danego środka transportu w pewnym obszarze były o 15-40% gorsze dla zestawów opcji, które nie wykorzystywały wszystkich przyspieszeń (zestawy 2-6) oraz od 0% do 289% dla zapytań o połączenia przechodzące przez poszczególne stacje. Rozrzut skuteczności opcji był różny dla oby rodzajów zapytań. Bardziej podobne były między sobą rezultaty osiągnięte dla zapytań o stacje, natomiast zyski dla zapytań o połączenia były niskie dla małych dystansów, a znacząco większe dla większych dystansów. Warto także zauważyć ze ilość zapytań o połączenia nie zależy tylko od skuteczności reguł odrzucających przesiadki, ale również od tego jak dużo możliwości podziałów na etapy odrzuciły reguły doboru środków transportu.

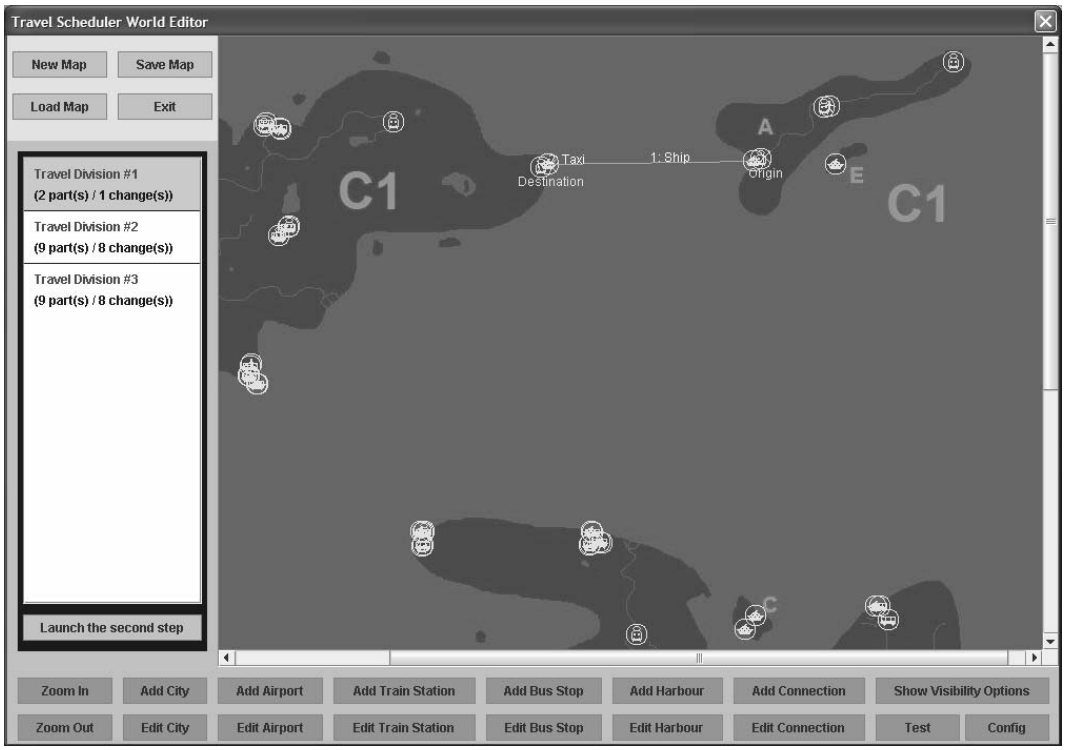
Na podstawie testów można stwierdzić, iż przygotowany mechanizm przyspieszający wyszukiwanie połączeń znacząco ogranicza ilość rozpatrywanych możliwości bez straty jakości odnajdowanych rozwiązań.

5 Demonstracja działania mechanizmów podnoszących jakość rozwiązań i efektywność algorytmu planującego

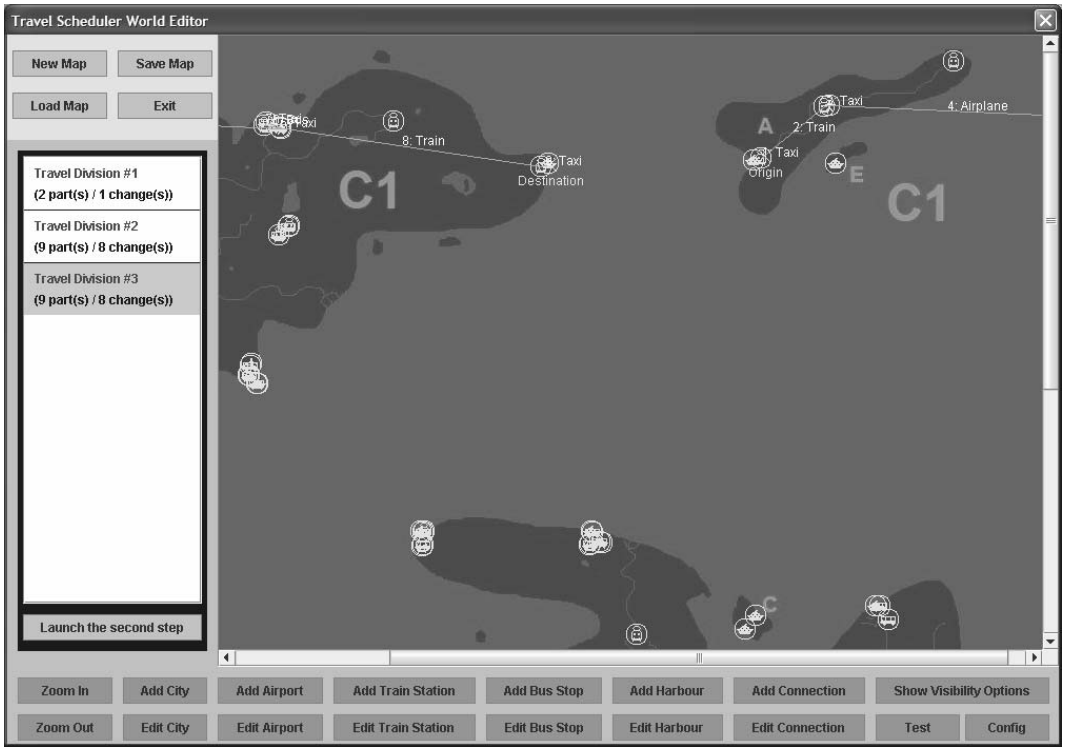
Niniejszy rozdział przedstawia działanie mechanizmów podnoszących jakość rozwiązań algorytmu planującego w trzech sytuacjach, w których jest możliwe zaobserwowanie efektów ich działania. Wiele z istniejących mechanizmów jest trudnych do obserwacji w sposób bezpośredni, na przykład, trudno jest obserwować działanie reguł odrzucających przesiadki inaczej niż poprzez zliczanie zapytań o połączenia przechodzące przez daną stację.

5.1 Dzielenie podróży na etapy zgodnie z regułami

Na poniższych rysunkach widać, że wygenerowane podziały podróży pomiędzy dwoma lokacjami leżącymi na różnych wyspach są logiczne w tym sensie, że morze jest pokonywane tylko przy pomocy rejsu statkiem lub lotu samolotem. Algorytm nie dopuścił do szukania, na przykład, połączenia kolejowego pomiędzy początkiem i końcem podróży. Trzecia nieprzedstawiona na rysunku propozycja jest podobna do tej z lotem samolotem – inna trasa prowadzi do lotniska na wyspie po lewej.



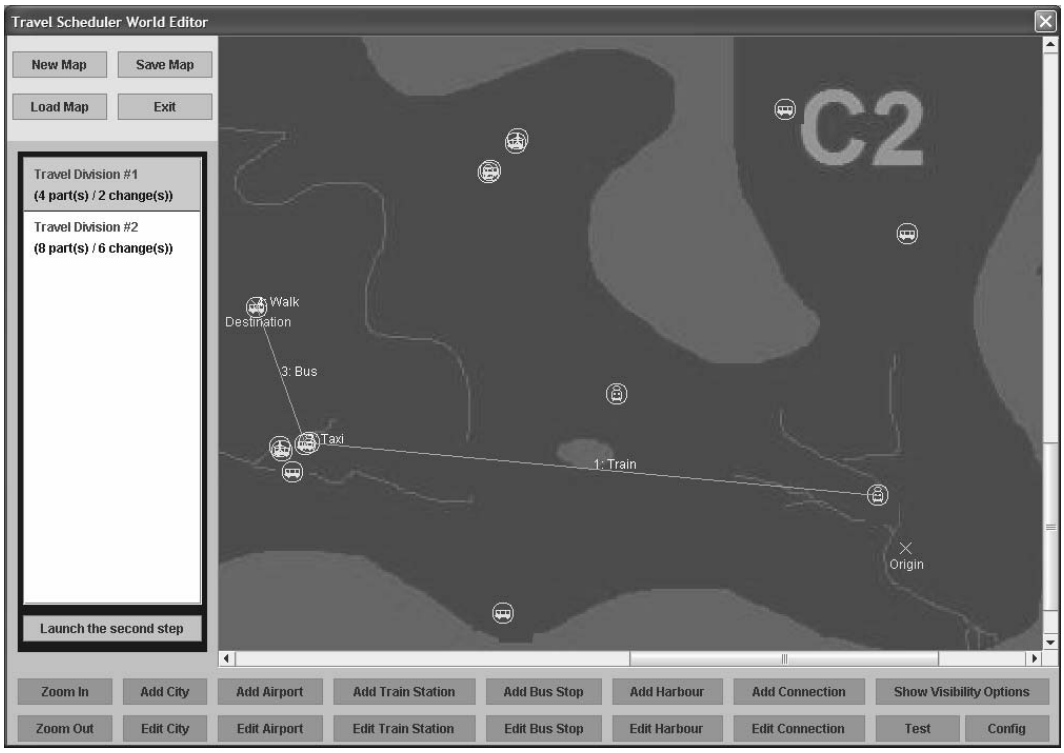
Rysunek 28 Podział na etapy z wykorzystaniem rejsu



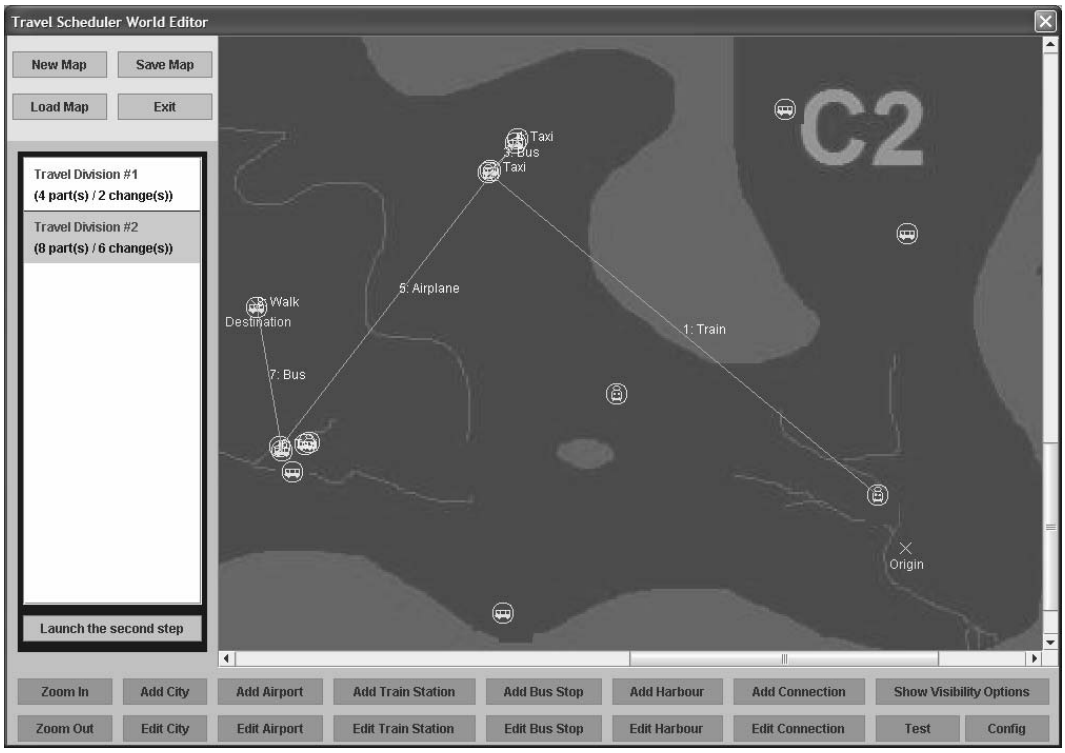
Rysunek 29 Podział na etapy z wykorzystaniem lotu samolotem

5.2 Minimalizacja niezaplanowanego dystansu

Na poniższych rysunkach widać, że w momencie, gdy w okolicy początku podróży nie ma żadnych stacji następuje minimalizacja niezaplanowanych odcinków. Z przygotowanych propozycji pozostały tylko te, które zaczynają się najbliżej początku podróży. Wszystkie pozostałe możliwe plany musiałyby się zaczynać od stacji leżących znacznie dalej niż ta najbliższa.



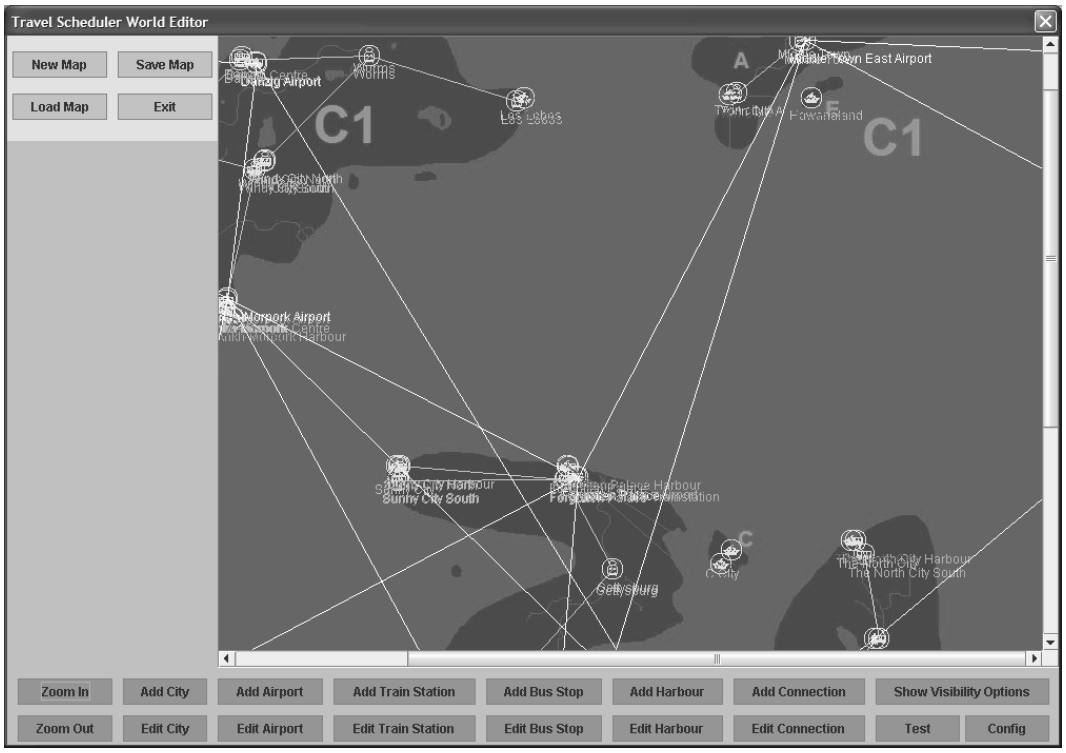
Rysunek 30 Jedna z pozostawionych propozycji podróży po minimalizacji niezaplanowanych dystansów



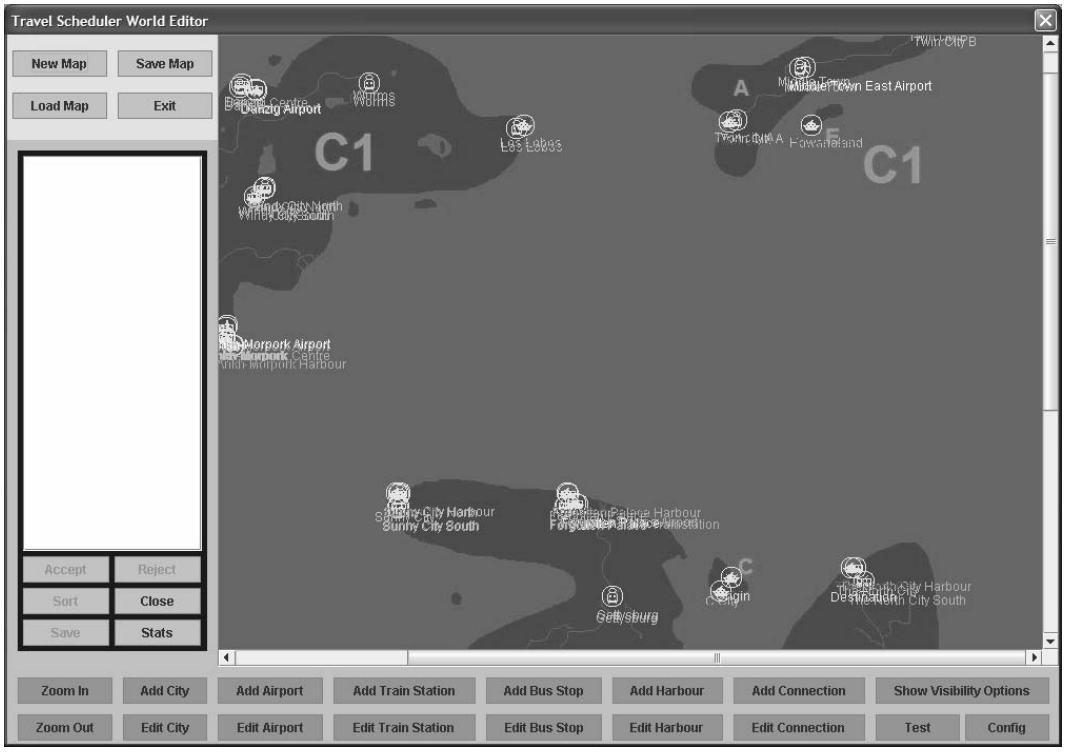
Rysunek 31 Jedna z pozostawionych propozycji podróży po minimalizacji niezaplanowanych dystansów

5.3 Działanie wiedzy eksperckiej

Na poniższym rysunku widać, że z wyspy C można wypłynąć tylko statkiem. Z portu leżącego na północy tej wyspy wypływa tylko jeden rejs na wyspę A. W momencie, gdy będzie planowana podróż z miejsca leżącego w pobliżu tego północnego portu do miejsca leżącego obok portu na wschód od wyspy, to w drugim etapie planowania reguły przesiadek odrzucą możliwość rejsu na północ i potem na zachód i dopiero stamtąd na południowy-wschód ze względu na to, że pierwsze dwa połączenia nie prowadzą w kierunku celu podróży. Oznacza to, że nie zostanie znaleziony żaden plan z braku możliwych połączeń.

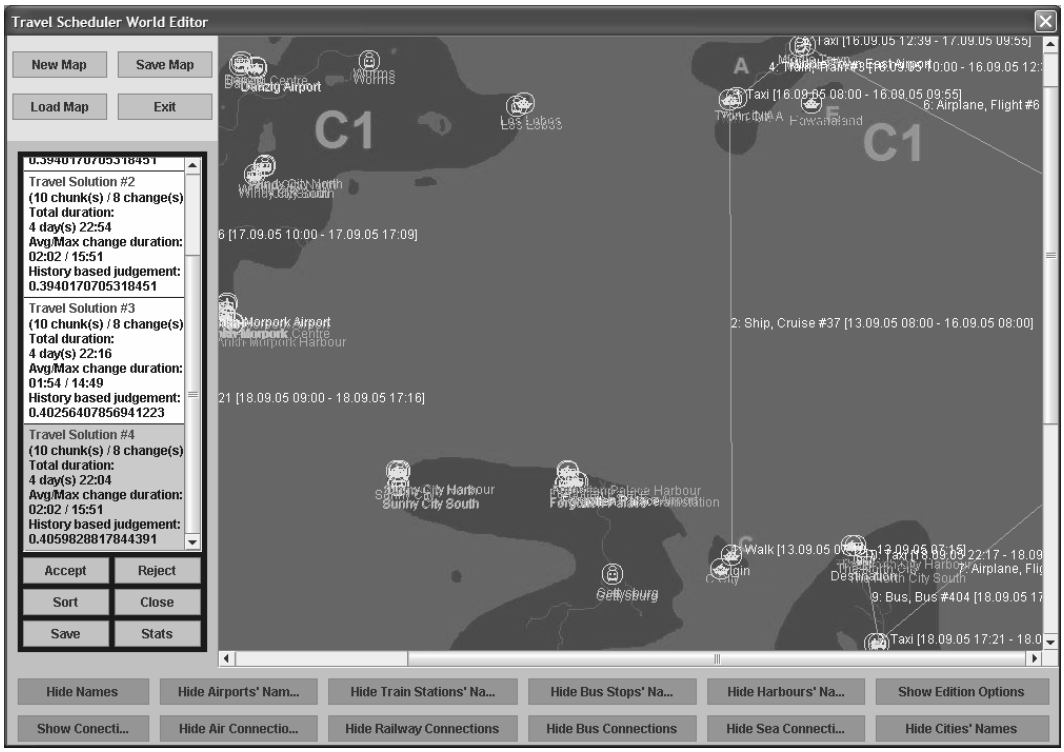


Rysunek 32 Z wyspy C można wypłynąć z portu północnego tylko rejsem na północ



Rysunek 33 Brak znalezionych rozwiązań

Tej sytuacji można jednak przeciwdziałać wykorzystując mechanizm dodawania wiedzy eksperckiej. Wystarczy dodać parę portów lotniczych: jeden (początkowy) z wyspy na północy mapy, a drugi (docelowy) leżący na południowym-wschodzie od źródłowego portu morskiego. W tym momencie algorytm będzie dodatkowo rozpatrywał podziały na etapy zawierające połączenia lotnicze z północy na południe. Dzięki temu II etap algorytmu planowania będzie mógł znaleźć plan prowadzący najpierw rejsem na północ, a następnie na ląd docelowy drogą lotniczą.



Rysunek 34 Odnalezione połączenie z wykorzystaniem połączenia lotniczego z północy na południe

Podsumowanie

Niniejsza praca przedstawiła zagadnienia związane z systemem automatycznego planowania podróży. Zostały omówione zarówno kwestie teoretyczne dotyczące głównie różnych pomysłów na sposób działania takiego systemu i postulatów dotyczących jego funkcjonalności oraz sposobów interakcji z użytkownikiem. Został także przedstawiony - w części praktycznej pracy - algorytm planowania podróży między dwoma dowolnymi punktami wykorzystujący różne środki transportu. Stanowi on krok do przodu w stosunku do aktualnie istniejących rozwiązań z dziedziny turystyki. W połączeniu z innymi elementami wspólnego projektu umożliwi on stworzenie kompleksowego systemu, który będzie w stanie pomóc zarówno osobie chcącej udać się na wyjazd turystyczny do odległego kraju, jak i osobie, która potrzebuje porady jak dotrzeć do najbliższej restauracji. System taki będzie musiał gromadzić olbrzymie ilości różnorodnych danych. Aby zapanować nad tymi danymi będzie musiał korzystać z jasno zdefiniowanego sposobu ich opisu, jaki stanowią ontologie. Zarówno dziedzina turystyki, jak i wiele innych, dąży powoli do stworzenia sieci semantycznej zdolnej do rozumienia przetwarzanych danych, właśnie dzięki zastosowaniu ontologii. Ciągle jednak dla wielu dziedzin nie powstały jednolite ontologie, dla wielu z nich takie ontologie są ciągle tworzone – powstają nawet specjalne organizacje, których celem jest realizacja tego celu. Zanim jednak zostanie stworzona taka sieć minie jeszcze trochę czasu. Należy go wykorzystać, aby gdy ona już powstanie, inne technologie były gotowe do wykorzystania w pełni jej potencjału. Należy także pamiętać, że twórcom ontologii jest łatwiej nad nimi pracować, gdy widzą dla nich realne zastosowania – jest to dla nich bodziec do dalszego działania.

Spis rysunków

Spis tabel

Bibliografia

[1] „An Agent-Based Architecture for Wireless Bus Travel Assistants", Strahan, R.; Muldoon, C.; O'Hare, G.M.P.; Bertolotto, M.; Collier, R.W. [on-line]. [dostęp: 15 października 2004] Dostępny w World Wide Web: http://emc2.ucd.ie/publications/WIS03.pdf

[2] Barish, Greg; Knoblock, Craig A., „Learning Efficient Value Predictors for Speculative Plan Execution" [on-line]. [dostęp: 10 września 2004] Dostępne w World Wide Web: http://www.isi.edu/info-agents/papers/barish02-webdb.pdf

[3] Barish, Greg; Knoblock, Craig A., „Learning value predictors for the speculative execution of information gathering plans" [on-line]. [dostęp: 10 września 2004] Dostępne w World Wide Web: http://www.isi.edu/info-agents/papers/barish03-ijcai.pdf

[4] Barish, Greg; Knoblock, Craig A., „Speculative execution for information gathering plans" [on-line]. [dostęp: 10 września 2004] Dostępne w World Wide Web: http://www.isi.edu/info-agents/papers/barish02-aips.pdf

[5] Barish, Greg; Knoblock, Craig A.; Minton, Steven, „Speculative execution for information agents" [on-line]. [dostęp: 10 września 2004] Dostępne w World Wide Web: http://www.isi.edu/info-agents/papers/barish00-aaai.pdf

[6] Barish, Greg; Knoblock, Craig A.; Minton, Steven, „Speculative execution for information agents" [on-line]. [dostęp: 10 września 2004] Dostępne w World Wide Web: http://www.isi.edu/info-agents/papers/barish00-aaai.pdf

[7] Cheyer, Adam; Julia, Luc, „Multimodal Maps: An Agent-Based Approach" [on-line]. [dostęp: 23 września 2004] Dostępne w World Wide Web: http://www.adam.cheyer.com/papers/lnai1374.pdf

[8] „An Agent-Mediated E-Commerce Framework for Tourism Domain", Dikenelli, Oguz; Topaloglu, N. Yasemin; Erdur, Cenk; Ünalir, Osman [on-line]. [dostęp: 29 sierpnia 2004] Dostępne w World Wide Web: http://www.srdc.metu.edu.tr/webpage/

projects/hermesProject/documents/AGENT-TOURISM.PDF

[9] Dillenburg, John F.; Wolfson, Ouri; Nelson, Peter C., „The Intelligent Travel Assistant" [on-line]. [dostęp: 12 września 2004] Dostępny w World Wide Web: http://www.cs.uic.edu/~wolfson/mobile_ps/ita02.pdf

[10] Edelkamp, Stefan; Jabbar, Shahid; Willhalm, Thomas, "Geometric Travel Planning" [on-line]. [dostęp: 2 października 2004] Dostępne w World Wide Web: http://ls5-www.cs.uni-dortmund.de/~jabbar/publications/ GeometricTravelPlanning.pdf

[11] „FIPA Personal Travel Assistance Specification" [on-line]. [dostęp: 15 lipca 2004] Dostępny w World Wide Web: http://www.fipa.org/specs/fipa00080/XC00080B.pdf

[12] Gambardella, Luca Maria; Rizzoli, Andrea E., „Agent-based Planning and Simulation of Combined Rail/Road Transport" [on-line]. [dostęp: 19 listopada 2004] Dostępny w World Wide Web: http://www.idsia.ch/~luca/simulation02.pdf

[13] „Getting from here to there: Interactive planning and agent execution for optimizing travel", Ambite, Jose Luis; Barish, Greg; Knoblock, Craig A.; Muslea, Maria; Oh, Jean; Minton, Steven [on-line]. [dostęp: 10 września 2004] Dostępne w World Wide Web: http://www.isi.edu/info-agents/papers/ambite02-iaai.pdf

[14] Gordon, M.; Paprzycki, M., „Designing Agent Based Travel Support System" [on-line]. [dostęp: 25 sierpnia 2005] Dostępne w World Wide Web: http://www.cs.okstate.edu/~marcin/mp/cvr/research/ ISPDC_2005.pdf

[15] Grundy, John; Jin, Weiguo, "Experiences developing a thin-client, multi-device travel planning application" [on-line]. [dostęp: 10 września 2004] Dostępne w World Wide Web: http://www.cs.auckland.ac.nz/~john-g/papers/chinz2002.pdf

[16] Jaric, Peter, "An Agent-Based Location System" [on-line]. [dostęp: 10 września 2004] Dostępne w World Wide Web: http://www.jaric.org/thesis.pdf

[17] Kambhampati, Subbarao; Mali, Amol; Srivastava, Biplav, "Hybrid Planning for Partially Hierarchical Domains" [on-line]. [dostęp: 11

września 2004] Dostępne w World Wide Web: http://rakaposhi.eas.asu.edu/aaai-98.pdf

[18] Kay, Michael G.; Jain, Ashish, „Issues in Agent-based Coordination of Public Logistics Networks" [on-line]. [dostęp: 24 września 2004] Dostępny w World Wide Web: http://www.ie.ncsu.edu/kay/pln/IETR02-01.pdf

[19] Kozłowski, Krzysztof; Dutkiewicz, Piotr; Wróblewski, Waldemar, „Planowanie zadań i programowanie robotów", Wydawnictwo Politechniki Poznańskiej, Poznań 1999.

[20] Kumar, Praveen; Reddy, Dhanunjaya; Singh, Varun, "Intelligent transport system using GIS" [on-line]. [dostęp: 25 września 2004] Dostępne w World Wide Web: http://www.gisdevelopment.net/application/Utility/ transport/pdf/164.pdf

[21] „Mixed-initiative, multi-source information assistants", Knoblock, Craig A.; Minton, Steven ; Ambite, Jose Luis; Muslea, Maria; Oh, Jean; Frank, Martin [on-line]. [dostęp: 10 września 2004] Dostępne w World Wide Web: http://www.isi.edu/info-agents/papers/knoblock01-www.pdf

[22] Nau, Dana S., „Automated Planning: Theory and Practice", rozdział 11: „Hierarchical Task Network Planning" [slajdy do wykładów] [on-line]. [dostęp: 1 listopada 2004] Dostępne w World Wide Web: http://www.cs.umd.edu/~nau/cmsc722/notes-fall-2004/chapter11.pdf

[23] Nau, Dana S., „Ordered Task Decomposition: Theory and Practice" [slajdy] [on-line]. [dostęp: 1 listopada 2004] Dostępne w World Wide Web: http://prometeo.ing.unibs.it/sschool/slides/nau/nau1.ppt

[24] Nau, Dana S.; Smith, Stephen J. J.; Erol, Kutluhan, „Control Strategies in HTN Planning: Theory Versus Practice" [on-line]. [dostęp: 1 listopada 2004] Dostępny w World Wide Web: http://www.cs.umd.edu/~nau/papers/planning-iaai98.pdf

[25] Nigarnjanagool, Suphasawas, „Development and Evaluation of an Agent-Based Adaptive Traffic Signal Control System" [on-line]. [dostęp: 6 listopada 2004] Dostępny w World Wide Web: http://www-civil.eng.monash.edu.au/people/centres/its/ WorkshopsSeminars/CAITRHomePage/CAITR_PREVIOUS_

PROCEEDINGS/CAITR_2003/nigarnjanagool.pdf

[26] Peuret, Frederic, „Case-Based Travel Agent” [on-line]. [dostęp: 28 sierpnia 2004] Dostępne w World Wide Web: http://www.cs.tcd.ie/publications/tech-reports/reports.99/TCD-CS-1999-69.pdf

[27] „SHOP2: An HTN Planning System” Nau, D. S.; Au, T.C.; Ilghami, O.; Kuter, U.; Murdock, J.W.; Wu, D.; Yaman, F. [on-line]. [dostęp: 1 listopada 2004] Dostępne w World Wide Web: http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume20/nau03a.pdf

[28] Stallard, David, "Talk'n'Travel: A Conversational System for Air Travel Planning" [on-line]. [dostęp: 29 sierpnia 2004] Dostępne w World Wide Web: http://acl.ldc.upenn.edu/A/A00/A00-1010.pdf

[29] „Trip-planner: An Agent Framework for Collaborative Trip Planning", Homb, Andrew; Mundhe, Manisha; Kimsen, Sonali; Sen, Sandip [on-line]. [dostęp: 1 września 2004] Dostępne w World Wide Web: http://www.cs.wright.edu/people/faculty/mcox/mii/papers/manisha.pdf

[30] Tuchinda, Rattapoom; Knoblock, Craig A., „Agent Wizard: Building Information Agents by Answering Questions” [on-line]. [dostęp: 10 września 2004] Dostępne w World Wide Web: http://www.isi.edu/info-agents/papers/tuchinda04-iui.pdf

[31] Waszkiewicz, Paweł; Cunningham, Padraig; Byrne, Ciara, „Case-based User Profiling in a Personal Travel Assistant” [on-line]. [dostęp: 29 sierpnia 2004] Dostępne w World Wide Web: http://www.cs.usask.ca/UM99/Proc/short/WaszkiewiczP.pdf

[32] Automapa [on-line]. [dostęp: 20 lipca 2005] Dostępna w World Wide Web: http://www.automapa.com.pl/

[33] Chef Moz [on-line]. [dostęp: 10 sierpnia 2005] Dostępny w World Wide Web: http://www.chefdmoz.com/

[34] Ebookers [on-line]. [dostęp: 19 lipca 2005] Dostępne w World Wide Web: http://www.ebookers.com/

[35] Eclipse [on-line]. [dostęp: 11 września 2005] Dostępny w World Wide Web: http://www.eclipse.org/

[36] Getty Thesaurus of Geographic Names [on-line]. [dostęp: 5 lutego 2005]

Dostępny w World Wide Web: http://www.getty.edu/research/conducting_research/vocabularies/tgn/

[37] Google Maps [on-line]. [dostęp: 11 września 2005] Dostępne w World Wide Web: http://maps.google.com/

[38] Heracles: Constrain-based Integration [on-line]. [dostęp: 10 września 2004] Dostępne w World Wide Web: http://www.isi.edu/info-agents/Heracles/

[39] Java Expert System Shell [on-line]. [dostęp: 5 września 2005] Dostępny w World Wide Web: http://herzberg.ca.sandia.gov/jess/

[40] Opodo [on-line]. [dostęp: 19 lipca 2005] Dostępne w World Wide Web: http://www.opodo.co.uk/

[41] Polskie Koleje Państwowe [on-line]. [dostęp: 6 sierpnia 2005] Dostępne w World Wide Web: http://www.pkp.pl/

[42] Resource Description Framework [on-line]. [dostęp: 11 września 2005] Dostępne w World Wide Web: http://www.w3.org/RDF/

[43] Theseus: Plan Execution [on-line]. [dostęp: 10 września 2004] Dostępne w World Wide Web: http://www.isi.edu/info-agents/Theseus/

[44] Traveliada [on-line]. [dostęp: 19 lipca 2005] Dostępna w World Wide Web: http://www.traveliada.pl/

[45] Travelocity [on-line]. [dostęp: 10 września 2004] Dostępne w World Wide Web: http://www.travelocity.co.uk/

[46] Web Ontology Language [on-line]. [dostęp: 11 września 2005] Dostępne w World Wide Web: http://www.w3.org/TR/owl-features/

Załączniki I: Pseudokod algorytmu generującego podział podróży na etapy

Poniżej został przedstawiony pseudokod I etapu algorytmu planowania podróży. Rozwiązywanie problemu rozpoczyna się wywołaniem:

```
GenerujPodział(lokacjaStartowa, lokacjaDocelowa, NULL, NULL, 1);
```

A oto jego implementacja:

```
GenerujPodział(początek, koniec, stacjaPoczątkowa, stacjaKońcowa, głębokość)
{
    jeżeli (osiągnięto maksymalna głębokość), to zwróć pustą listę;
    odległość ← ObliczOdległość(początek, koniec);
    środkiTransportu ← WybierzŚrodkiTransportu(początek, koniec);
    dla każdego środka transportu z listy środkiTransportu
    {
        środekTransportu ← następny środek transportu z listy;
        // wybierz stacje
        stacjePoczątkowe ← WybierzStacje(środekTransportu , początek, ε *
        odległość);
        stacjeKońcowe ← WybierzStacje(środekTransportu , koniec, ε * odległość);
        // rozwiąż każdą z nich (operacje te można wykonać równolegle)
        dla każdej stacji z listy stacjePoczątkowe
        {
            stacja ← następna stacja z listy;
            jeśli (stacja == stacjaPoczątkowa lub stacja == stacjaKońcowa), to
            kontynuuj pętlę;
            dodaj stację do listy wykorzystaneStacjePoczątkowe;
            oznacz stację do parowania z każdą zwykłą docelową;
            dojazdyPoczątkowe(stacja) ← GenerujPodział(początek,
            stacja.lokalizacja, stacjaPoczątkowa, stacja, głębokość + 1);
```

```
}
dla każdej stacji z listy stacjeKońcowe
{
    stacja ← następna stacja z listy;
    jeśli (stacja == stacjaPoczątkowa lub stacja == stacjaKońcowa), to
    kontynuuj pętlę;
    dodaj stację do listy wykorzystaneStacjeKońcowe;
    oznacz stację do parowania z każdą zwykłą początkową;
    dojazdyKońcowe(stacja) ← GenerujPodział(stacja.lokalizacja, koniec,
    stacja, stacjaKońcowa, głębokość + 1);
}
// dodaj wiedzę ekspercką
jeśli (głębokość <= limitGłębokościNaWiedzęEkspercką), to
{
    eksperckieParyStacji ← PodajEksperckieParyStacji(początek, koniec,
    środekTransportu);
    dla każdej pary z listy eksperckieParyStacji
    {
        para ← następna para z listy;
        jeśli (para.stacjaPoczątkowa == stacjaPoczątkowa lub
        para.stacjaPoczątkowa == stacjaKońcowa), to kontynuuj pętlę;
        jeśli (para.stacjaKońcowa == stacjaPoczątkowa lub
        para.stacjaKońcowa == stacjaKońcowa), to kontynuuj pętlę;
        jeśli (para.stacjaPoczątkowa jest na liście
        wykorzystaneStacjePoczątkowe oraz para.stacjaKońcowa jest na
        liście wykorzystaneStacjeKońcowe), to kontynuuj pętlę;
        oznacznik ← WygenerujNowyOznacznik();
        dodaj para.stacjaPoczątkowa do listy
        wykorzystaneStacjePoczątkowe;
        oznacz para.stacjaPoczątkowa do parowania tylko z tą drugą z
        pary;
        dojazdyPoczątkowe(para.stacjaPoczątkowa) ←
        GenerujPodział(początek, para.stacjaPoczątkowa.lokalizacja,
```

```
                    stacjaPoczątkowa, para.stacjaPoczątkowa, głębokość + 1);
                    dodaj para.stacjaKońcowa do listy wykorzystaneStacjeKońcowe;
                    oznacz para.stacjaKońcowa do parowania tylko z tą drugą z pary;
                    dojazdyKońcowe(para.stacjaKońcowa) ← GenerujPodział(para.
                    stacjaKońcowa.lokalizacja, koniec, para.StacjaKońcowa,
                    stacjaKońcowa, głębokość + 1);
                }
                // budowanie podziału
                dla każdej stacji z listy wykorzystaneStacjePoczątkowe
                {
                    wykorzystanaStacjaPoczątkowa ← następna stacja z listy
                    wykorzystaneStacjePoczątkowe;
                    podziałyPoczątkowe ←
                    dojazdyPoczątkowe(wykorzystanaStacjaPoczątkowa);
                    jeśli (podziałyPoczątkowe.rozmiar == 0), to dodaj pusty podział;
                    dla każdej stacji z listy wykorzystaneStacjeKońcowe
                    {
                        wykorzystanaStacjaKońcowa ← następna stacja z listy
                        wykorzystaneStacjeKońcowe;
                        jeśli (wykorzystanaStacjaPoczątkowa ==
                        wykorzystanaStacjaKońcowa), to przejdź do kolejnej
                        wykorzystanej stacji końcowej;
                        jeśli (wykorzystanaStacjaPoczątkowa i
                        wykorzystanaStacjaKońcowa nie są oznaczone do
                        wzajemnego sparowania), to przejdź do kolejnej
                        wykorzystanej stacji końcowej;
                        jeśli (stacje wykorzystanaStacjaPoczątkowa i
                        wykorzystanaStacjaKońcowa nie są stacjami eksperckimi),
                        to
                            jeśli (połączenie między nimi jest wbrew regułom), to
                            przejdź do kolejnej wykorzystanej stacji końcowej;
                        // można tworzyć podział
                        głównyEtap ← StwórzNowyEtap(środekTransportu,
```

```
                    wykorzystanaStacjaPoczątkowa,
                    wykorzystanaStacjaKońcowa);
                    dla każdego podziału z listy podziałyPoczątkowe
                    {
                        podziałPoczątkowy ← następny podział z listy;
                        jeśli (podziałPoczątkowy.IlośćEtapów() > 0), to
                            jeśli (ObliczOdległość(podziałPoczątkowy.
                            ostatniaStacja.lokacja,
                            wykorzystanaStacjaPoczątkowa.lokacja) jest
                            zbyt duża), to przejdź do następnego
                            podziału początkowego;
                        kopia ← kopia podziałPoczątkowy;
                        dodaj do kopii głównyEtap;
                        jeśli (kopia.IlośćPrzesiadek() jest większa niż limit),
                        to przejdź do następnego podziału początkowego;
                        dodaj kopię do podziałyZGłównymEtapem;
                    }
                    jeśli (podziałyZGłównymEtapem.rozmiar == 0), to przejdź
                    do kolejnej wykorzystanej stacji końcowej;
                    // dodawanie podziałów końcowych
                    podziałyKońcowe ←
                    dojazdyKońcowe(wykorzystanaStacjaKońcowa);
                    jeśli (podziałyKońcowe.rozmiar == 0), to dodaj pusty
                    podział;
                    dla każdego podziału końcowego z listy podziałyKońcowe
                    {
                        podziałKońcowy ← następny podział z listy;
                        dla każdego podziału z listy
                        podziałyZGłównymEtapem
                        {
                            podziałZEtapemGłównym ← następny podział
                            z listy;
                            jeśli (podziałKońcowy.IlośćEtapów() > 0), to
```

```
                                    jeśli (ObliczOdległość(podziałKońcowy.
                                    pierwszaStacja.lokacja,
                                    podziałZEtapemGłównym.ostatniaStacja.
                                    lokacja) jest zbyt duża), to przejdź do
                                    następnego podziału z etapem głównym;
                                kopia ← kopia podziałZEtapemGłównym;
                                dodaj podziałKońcowy do kopii;
                                jeśli (kopia.IlośćPrzesiadek() jest większa niż
                                limit), to przejdź do następnego podziału z
                                etapem głównym;
                                // końcowe tworzenie podziału
                                odległośćPoczątkowa ←
                                ObliczOdległość(początek,
                                kopia.pierwszaStacja.lokacja);
                                odległośćKońcowa ←
                                ObliczOdległość(kopia.ostatniaStacja.lokacja,
                                koniec);
                                jeśli (odległość > odległośćPoczątkowa +
                                odległośćKońcowa), to dodaj kopię do
                                listaWynikowa;
                            }
                        }
                    }
                }
            }
        }
    }
    // usuwanie rozwiązań słabych jakościowo
    wybierz z listaWynikowa tylko te podziały, które zaczynają się blisko początku i
    kończą blisko końca, a gdy nie ma takich to te, które spełniają ten warunek
    przynajmniej z jednej strony, a gdy nie ma i tych, to weź wszystkie;
    usuń z listaWynikowa podziały, które mają odległości między początkiem i pierwszą
    stacją oraz między końcem i ostatnią stacją znacznie dłuższe niż inne (minimalizuj
    każdy koniec oddzielnie lub oba jednocześnie, zależnie od rodzaju podziałów z
```

```
        poprzedniego wyboru);
        usuń z listaWynikowa podziały, które są znacznie dłuższe niż inne (ale bez tych
        eksperckich);
        // te mogły się pojawić, gdy nie dało się inaczej w poszczególnych etapach,
        // ale może są inne podziały już bez nich
        usuń z listaWynikowa podziały, które zawierają niedopuszczalne środki transportu;
        zwróć listaWynikowa;
}
```

Załącznik II: Pseudokod algorytmu implementującego etapy podróży konkretnymi połączeniami

Zostanie teraz przedstawiony pseudokod algorytmu implementującego etapy podróży realnymi połączeniami. Listę możliwych implementacji zwróci wywołanie:

```
ZnajdźPołączenia(listaPodziałówWygenerowanaWEtapie1, ograniczenia);
```

A oto jego implementacja:

```
ZnajdźPołączenia(listaPodziałów, ograniczenia)
{
    jeśli (planowanie z datą wyjazdu), to
    {
        limitWyjazdu ← ograniczenia.dataWyjazdu;
        limitDotarcia ← NULL;
        odwracanie ← FAŁSZ;
    }
    w przeciwnym przypadku
    {
        // te przypisania są dobre, ponieważ w tym przypadku będziemy wszystko
        odwracać
        limitWyjazdu ← ograniczenia.dataGraniczna Dojazdu;
        limitDotarcia ← teraz;
        odwracanie ← PRAWDA;
    }
    dla każdego podziału z listy listaPodziałów
    {
        podział ← następny podział z listy;
        jeśli (odwracanie == PRAWDA), to podział ←
        podział.PodajOdwróconyPodział();
```

```
            ImplementujPodział(podział, limitWyjazdu, limitDotarcia, listaWynikowa);
        }
        UsuńImplementacjeŁamiąceOgraniczenia(listaWynikowa);
        jeśli (odwracanie == PRAWDA), to odwróć implementacje z listy listaWynikowa;
        UsuńPodobneImplementacje(listaWynikowa);
        zwróć listaWynikowa;
}
```

```
ImplementujPodział(podział, limitWyjazdu, limitDotarcia, listaWynikowa)
{
        // na tej liście będą zadania implementujące wszystkie dotychczasowe etapy
        zadania ← NowaListaZadań();
        // środek transportu bez rozkładu jazdy, to na przykład spacer lub taksówka
        zainicjowanoEtapZeŚrodkiemTransportuBezRozkładuJazdy ← FAŁSZ;
        dla każdego etapu z listy podział.etapy
        {
                etap ← następny etap z listy;
                jeśli (etap.środekTransportu jest środkiem transportu bez rozkładu jazdy), to
                {
                        odcinek ← NowyOdcinek(etap.środekTransportu, etap.początek,
                        etap.koniec);
                        odcinek.UstawStacje(etap.stacjaPoczątkowa, etap.stacjaKońcowa);
                        oznacz datę końcową odcinka do wyznaczenia potem;
                        jeśli (zadania.rozmiar > 0), to
                        {
                                // jest to np. spacer z przystanku autobusowego na dworzec
                                kolejowy
                                dla każdego zadania z listy zadania
                                {
                                        zadanie ← następne zadanie z listy;
                                        kopia ← kopia odcinek;
                                        ustaw datę początkową kopii na datę końcową
                                        zadanie.ostatniOdcinek;
```

```
                    dodaj kopię do zadanie;
                    zadanie.aktualnaStacja ← kopia.stacjaKońcowa;
                    zadanie.startowaStacja ← kopia.stacjaKońcowa;
                    zadania.połączenie ← NULL;
                    jeśli (zadanie.ilośćPrzesiadek > limit), to usuń zadanie z
                    zadania;
                }
                jeśli (zadania.rozmiar == 0), to wyjdź;
            }
            w przeciwnym przypadku
            {
                // jest to pierwszy etap w podróży, np. spacer z domu na autobus
                zadania ← NoweZadanie();
                oznacz datę początkową zadania do ustalenia potem;
                dodaj odcinek do zadanie;
            }
        }
        w przeciwnym przypadku
        {
            // jest to środek transportu z rozkładem jazdy, np. pociąg lub samolot
            // menadżer zadań działa jak stos
            menadżerZadań ← NowyMenadżerZadań();
            jeśli (zainicjowanoEtapZeŚrodkiemTransportuBezRozkładuJazdy ==
            PRAWDA), to
            {
                // kontynuacja zadań implementujących poprzednie etapy
                dla każdego zadania z listy zadania
                {
                    zadanie ← następne zadanie z listy zadania;
                    odcinekZUstawionąDatąKońcową ←
                    zadanie.ostatniOdcinekZUstawionąDatąKońcową;
                    jeśli (odcinekZUstawionąDatąKońcową jest NULL), to
                    przejdź do następnego zadania;
```

```
                    czasPrzesiadki ←
                    odcinekZUstawionąDatąKońcową.dataKońcowa;
                    // ewentualna rezerwacja czasu na np. spacer lub taksówkę
                    jeśli (ostatni odcinek w zadanie jest pokonywany środkiem
                    transportu bez rozkładu jazdy), to
                    PrzesuńCzasPrzesiadki(czasPrzesiadki,
                    zadanie.ostatniOdcinek.środekTransportu);
                    // możliwe przesiadki w nowy środek transportu (preferuj
                    bezpośrednie)
                    przesiadki ← WybierzPrzesiadki(etap.środekTransportu,
                    etap.stacjaPoczątkowa, etap.stacjaKońcowa,
                    czasPrzesiadki, limitDotarcia);
                    jeśli (przesiadki.rozmiar == 0, ale były przesiadki ze zbyt
                    długim czasem oczekiwania), to przesiadki ←
                    WybierzNPrzesiadek(etap.środekTransportu,
                    etap.stacjaPoczątkowa, etap.stacjaKońcowa,
                    czasPrzesiadki, limitDotarcia);
                    dla każdej przesiadki z listy przesiadki
                    {
                        przesiadka ← następna przesiadka z listy;
                        noweZadanie ← kopia zadanie;
                        zastosuj przesiadkę do noweZadanie;
                        jeśli (noweZadanie nie przekroczyło limitu
                        przesiadek), to
                        {
                            wyczyść listę odwiedzonych stacji w
                            noweZadanie;
                            dodaj etap.stacjaPoczątkowa do odwiedzonych
                            stacji w noweZadanie;
                            dodaj noweZadanie do menadżerZadań;
                        }
                    }
                }
```

```
}
w przeciwnym przypadku
{
    // pierwsze połączenia w ogóle - inicjacja listy zadania
    czasPrzesiadki ← kopia limitWyjazdu;
    jeśli (zadania nie są puste), to
    {
        // był jakiś etap ze środkiem transportu bez rozkładu jazdy
        poprzednieZadanie ← zadania.pierwszeZadanie;
        ostatniOdcinek ← poprzednieZadanie.ostatniOdcinek;
        jeśli (ostatniOdcinek ma ustawioną datę końcową), to
            czasPrzesiadki ← ostatniOdcinek.dataKońcowa;
        w przeciwnym wypadku
            PrzesuńCzasPrzesiadki(czasPrzesiadki,
            ostatniOdcinek.środekTransportu);
    }
    połączeniaStartowe ←
    WybierzPołączeniaStartowe(etap.środekTransportu,
    etap.stacjaPoczątkowa, etap.stacjaKońcowa, czasPrzesiadki,
    limitDotarcia);
    zainicjowanoEtapZeŚrodkiemTransportuBezRozkładuJazdy ←
    (połączeniaStartowe.rozmiar > 0);
    dla każdego połączenia z listy połączeniaStartowe
    {
        połączenie ← następne połączenie z listy
        połączeniaStartowe;
        zadanie ← NoweZadanie(połączenie);
        jeśli (zadania.rozmiar == 0), to
        {
            dodaj etap.stacjaPoczątkowa do odwiedzonych stacji
            w zadanie;
            dodaj zadanie do menadżerZadań;
        }
```

```
                    w przeciwnym przypadku
                    {
                        dla każdego zadania z listy zadania
                        {
                            zadanie ← następne zadanie z listy;
                            noweZadanie ← kopia zadanie;
                            zastosuj połączenie jako przesiadkę do
                            noweZadanie;
                            jeśli (noweZadanie nie przekroczyło limitu
                            przesiadek), to
                            {
                                wyczyść listę odwiedzonych stacji w
                                noweZadanie;
                                dodaj etap.stacjaPoczątkowa do
                                odwiedzonych stacji w noweZadanie;
                                dodaj noweZadanie do menadżerZadań;
                            }
                        }
                    }
                }
            }
            // rozwiązanie przygotowanych zadań zebranych w menadżerze
            // zadania można rozwiązywać równolegle
            rozwiązaniaEtapu ← NowaLista();
            jak długo (menadżerZadań.ilośćZadań > 0)
            {
                zadanie ← menadżerZadań.PodajNastępneZadanie();
                jeśli (zadanie.aktualnaStacja nie jest ostatnią stacją w
                zadanie.aktualnePołączenie), to
                {
                    następnaStacja ←
                    zadanie.aktualnePołączenie.NastępnaStacja();
                    jeśli (jeszcze nie odwiedzono następnaStacja w tym
```

```
                    zadaniu), to
                    {
                        // przejście do następnej stacji
                        zadanie.aktualnaStacja ← następnaStacja;
                        dodaj następnaStacja do odwiedzonych stacji w
                        zadanie;
                        jeśli (następnaStacja == etap.stacjąKońcowa), to
                        {
                            // dotarcie do końca etapu
                            odcinek ←
                            NowyOdcinek(etap.środekTransportu,
                            zadanie.stacjaStartowa.lokacja,
                            zadanie.aktualnaStacja.lokacja);
                            odcinek.UstawStacje(zadanie.stacjaStartowa,
                            zadanie.aktualnaStacja);
                            ustaw daty startu i końca dla odcinek zgodnie z
                            rozkładem jazdy zadanie.aktualnePołączenie;
                            dodaj odcinek do zadanie;
                            dodaj zadanie do rozwiązaniaEtapu;
                            przejdź do następnego zadania z
                            menadżeraZadań;
                        }
                        // to nie koniec podróży
                        // sprawdź czy są możliwe przesiadki
                        jeśli (zadanie nie przekroczyło limitu przesiadek), to
                        {
                            czasPrzesiadki ← czas przyjazdu na
                            zadanie.aktualnaStacja;
                            przesiadki ←
                            WybierzPrzesiadki(etap.środekTransportu,
                            zadanie.aktualnaStacja, etap.stacjaKońcowa,
                            czasPrzesiadki, limitDotarcia);
                            jeśli (przesiadki.rozmiar == 0, ale były
```

przesiadki ze zbyt długim czasem oczekiwania),
to
{
PrzesuńCzasPrzesiadki(*czasPrzesiadki*, *ograniczenia*.minimalnyCzasPrzesiadki)**;**
przesiadki ← **WybierzPołączeniaStartowe** (*etap*.środekTransportu, *zadanie*.aktualnaStacja, *etap*.stacjaKońcowa, *czasPrzesiadki*, *limitDotarcia*)**;**
}
dla każdej przesiadki **z listy** *przesiadki*
{
przesiadka ← **następna** przesiadka **z listy;**
jeśli (*zadanie*.aktualnePołączenie == *przesiadka*.połączenie), **to** przejdź do następnej przesiadki z listy;
jeśli (*przesiadka* jest wbres regułom), **to** przejdź do następnej przesiadki z listy;
noweZadanie ← kopia *zadanie***;**
odcinek ← **NowyOdcinek**(*etap*.środekTransportu, *noweZadanie*.stacjaStartowa.lokacja, *noweZadanie*.aktualnaStacja.lokacja)**;**
odcinek.**UstawStacje**(*noweZadanie*.stacjaStartowa, *noweZadanie*.aktualnaStacja)**;**
ustaw daty startu i końca dla *odcinek* zgodnie z rokładem jazdy *noweZadanie*.aktualnePołączenie**;**
dodaj *odcinek* do *noweZadanie***;**
zastosuj *przesiadkę* do *noweZadanie***;**

```
                                        jeśli (noweZadanie nie przekroczyło limitu
                                        przesiadek), to dodaj noweZadanie do
                                        menadżerZadań;
                                    }
                                }
                                dodaj zadanie do menadżerZadań;
                            }
                        }
                    }
                    // zastąp zadania listą rozwiązaniaEtapu
                    zadania ← rozwiązaniaEtapu;
                    jeśli (zadania.rozmiar == 0), to wyjdź;
                }
            }
            ustal daty dla odcinków bez ustawionych dat dla zadań z listy zadania;
            dodaj wszystkie zadania z listy zadania do listy listaWynikowa;
}
```

Załącznik III: Interfejs modułu wiedzy eksperckiej

Poniżej został przedstawiony interfejs, który musi być implementowany przez klasę dostarczającą par stacji stanowiących wiedzę ekspercką wspomagającą pierwszy etap algorytmu planowania podróży:

```
package scheduler.modulesinterfaces;

import scheduler.TravelConstraints;
import world.Location;
import world.MeanOfTransportation;

public interface FeedbackTravelPartsProvider
{
        public ArrayList GetFeedbackTravelParts(Location origin, Location
                                        destination, int meanOfTransportation);

        public void SetTravelConstraints(TravelConstraints constraints);
}
```

Zwracana lista (typu ArrayList) musi zawierać instancje klasy FeedbackStationsPair opisującej pojedynczą parę zwracanych par stacji.

Załącznik IV: Interfejs modułu geograficznego

Poniżej został przedstawiony interfejs, który musi być implementowany przez klasę dostarczającą listę stacji określonego środka transportu leżących w określonym prostokątnym obszarze:

```
package scheduler.modulesinterfaces;

import java.util.ArrayList;

public interface LocationsTravelInfoProvider
{
    public ArrayList GetAirportsInRange(int minX, int minY, int maxX, int maxY);

    public ArrayList GetTrainStationsInRange(int minX, int minY, int maxX, int maxY);

    public ArrayList GetBusStopsInRange(int minX, int minY, int maxX, int maxY);

    public ArrayList GetHarboursInRange(int minX, int minY, int maxX, int maxY);
}
```

Załącznik V: Interfejs modułu reguł doboru odpowiednich środków transportu

Poniżej został przedstawiony interfejs, który musi być implementowany przez klasę dostarczającą listę odpowiednich środków transportu do pokonania podanego odcinka:

```
package scheduler.modulesinterfaces;

import java.util.ArrayList;
import world.Location;

public interface MeansOfTransportationProvider
{
    public ArrayList GetMeansOfTransportation(Location origin, Location destination);

    public static final double MIN_FLIGHT_DISTANCE = 50.0;

    public static final double MIN_CRUISE_DISTANCE = 5.0;

    public static final double MIN_BUS_DISTANCE = 1.0;

    public static final double MIN_TRAIN_DISTANCE = 2.0;

    public static final double MAX_WALK_DISTANCE = 1.0;

    public static final double MIN_TAXI_DISTANCE = 1.0;

    public static final double MAX_TAXI_DISTANCE = 10.0;
}
```

Zwracana lista (typu ArrayList) musi zawierać instancje klasy Integer enkapsulujące stałe oznaczające odpowiednie środki transportu.

Załącznik VI: Interfejs modułu dostarczający list połączeń dla podanej stacji

Poniżej został przedstawiony interfejs, który musi być implementowany przez klasę dostarczającą listę połączeń przechodzących przez podaną stację:

```
package scheduler.modulesinterfaces;

import java.util.ArrayList;
import world.Station;

public interface ConnectionsTravelInfoProvider
{
    public ArrayList GetConnectionsForStation(Station station, int meanOfTransportation);
}
```

Załącznik VII: Interfejs modułu reguł odrzucania przesiadek

Poniżej został przedstawiony interfejs, który musi być implementowany przez klasę decydującą czy należy odrzucić daną przesiadkę:

```
package scheduler.modulesinterfaces;

import scheduler.ConnectionsFinderTask;
import scheduler.TravelConstraints;
import scheduler.TravelPart;

public interface ChangesJudger
{
    public boolean ShouldDropChange(ConnectionsFinderTask task, ConnectionsFinderTask changeTask, TravelPart travelPart, boolean fRevert);

    public void SetTravelConstraints(TravelConstraints constraints);
}
```

Załącznik VIII: Interfejs modułu oceniającego propozycje podróży na podstawie historii

Poniżej został przedstawiony interfejs, który musi być implementowany przez klasę oceniającą propozycje podróży:

```
package scheduler.modulesinterfaces;

import scheduler.ConnectionsFinderTask;

public interface TravelSolutionsJudger
{
    public double JudgeTravelSolution(ConnectionsFinderTask travelSolution);

    public void AddTravelSolution(ConnectionsFinderTask travelSolution, boolean fAccepted);
}
```

Załącznik IX: Zapis reguł doboru odpowiednich środków transportu w języku JESS

```
 (defrule continentsAvailableRule1
        (continent1 ?&~NotAvailable)
        (continent2 ?&~NotAvailable)
=>
        (assert (continentsAvailable TRUE))
)

(defrule continentsAvailableRule2
        (continent1 ?&NotAvailable)
=>
        (assert (continentsAvailable FALSE))
)

(defrule continentsAvailableRule3
        (continent2 ?&NotAvailable)
=>
        (assert (continentsAvailable FALSE))
)

(defrule islandsAvailableRule1
        (island1 ?&~NotAvailable)
        (island2 ?&~NotAvailable)
=>
        (assert (islandsAvailable TRUE))
)

(defrule islandsAvailableRule2
        (island1 ?&NotAvailable)
=>
        (assert (islandsAvailable FALSE))
)

(defrule islandsAvailableRule3
        (island2 ?&NotAvailable)
=>
        (assert (islandsAvailable FALSE))
)

(defrule sameContinentRule1
        (continent1 ?x&~NotAvailable)
        (continent2 ?x)
=>
        (assert (sameContinent TRUE))
)
```

```
(defrule sameContinentRule2
	(continent1 ?x&~NotAvailable)
	(continent2 ?y&~NotAvailable&:(neq ?x ?y))
=>
	(assert (sameContinent FALSE))
)

(defrule sameIslandRule1
	(exists (sameContinent TRUE))
	(island1 ?x&~NotAvailable)
	(island2 ?x)
=>
	(assert (sameIsland TRUE))
)

(defrule sameIslandRule2
	(exists (sameContinent TRUE))
	(island1 ?x&~NotAvailable)
	(island2 ?y&~NotAvailable&:(neq ?x ?y))
=>
	(assert (sameIsland FALSE))
)

(defrule airplaneDistanceRule1
	(airplaneMinDistance ?x)
	(distance ?y&:(>= ?y ?x))
=>
	(assert (airplaneDistance TRUE))
)

(defrule airplaneDistanceRule2
	(airplaneMinDistance ?x)
	(distance ?y&:(< ?y ?x))
=>
	(assert (airplaneDistance FALSE))
)

(defrule airplaneRule1
	(exists (airplaneDistance TRUE))
=>
	(assert (properMean Airplane))
)

(defrule airplaneRule2
	(exists (airplaneDistance FALSE))
	(exists (sameContinent TRUE))
	(exists (islandsAvailable TRUE))
	(exists (sameIsland FALSE))
=>
```

```
        (assert (properMean Airplane))
)

(defrule airplaneRule3
        (exists (airplaneDistance FALSE))
        (exists (continentsAvailable TRUE))
        (exists (sameContinent FALSE))
=>
        (assert (properMean Airplane))
)

(defrule trainDistanceRule1
        (trainMinDistance ?x)
        (distance ?y&:(>= ?y ?x))
=>
        (assert (trainDistance TRUE))
)

(defrule trainRule1
        (exists (trainDistance TRUE))
        (exists (continentsAvailable FALSE))
=>
        (assert (properMean Train))
)

(defrule trainRule2
        (exists (trainDistance TRUE))
        (exists (sameContinent TRUE))
        (exists (islandsAvailable FALSE))
=>
        (assert (properMean Train))
)

(defrule trainRule3
        (exists (trainDistance TRUE))
        (exists (sameContinent TRUE))
        (exists (sameIsland TRUE))
=>
        (assert (properMean Train))
)

(defrule busDistanceRule1
        (busMinDistance ?x)
        (distance ?y&:(>= ?y ?x))
=>
        (assert (busDistance TRUE))
)

(defrule busRule1
        (exists (busDistance TRUE))
```

```
        (exists (continentsAvailable FALSE))
=>
        (assert (properMean Bus))
)

(defrule busRule2
        (exists (busDistance TRUE))
        (exists (sameContinent TRUE))
        (exists (islandsAvailable FALSE))
=>
        (assert (properMean Bus))
)

(defrule busRule3
        (exists (busDistance TRUE))
        (exists (sameContinent TRUE))
        (exists (sameIsland TRUE))
=>
        (assert (properMean Bus))
)

(defrule shipDistanceRule1
        (shipMinDistance ?x)
        (distance ?y&:(>= ?y ?x))
=>
        (assert (shipDistance TRUE))
)

(defrule shipRule1
        (exists (shipDistance TRUE))
=>
        (assert (properMean Ship))
)

(defrule walkDistanceRule1
        (walkMaxDistance ?x)
        (distance ?y&:(<= ?y ?x))
=>
        (assert (walkDistance TRUE))
)

(defrule walkRule1
        (exists (walkDistance TRUE))
=>
        (assert (properMean Walk))
)

(defrule taxiDistanceRule1
        (taxiMinDistance ?x)
        (taxiMaxDistance ?y)
```

```
        (walkMaxDistance ?z)
        (distance ?d&:(>= ?d ?x)&:(<= ?d ?y)&:(> ?d ?z))
=>
        (assert (taxiDistance TRUE))
)

(defrule taxiRule1
        (exists (taxiDistance TRUE))
        (exists (continentsAvailable FALSE))
=>
        (assert (properMean Taxi))
)

(defrule taxiRule2
        (exists (taxiDistance TRUE))
        (exists (sameContinent TRUE))
        (exists (islandsAvailable FALSE))
=>
        (assert (properMean Taxi))
)

(defrule taxiRule3
        (exists (taxiDistance TRUE))
        (exists (sameContinent TRUE))
        (exists (sameIsland TRUE))
=>
        (assert (properMean Taxi))
)
```

Załącznik X: Zapis reguł odrzucania przesiadek w języku JESS

```
(defrule directConnectionsRule1
        (exists (currentConnectionIsDirect TRUE))
        (exists (changeConnectionIsDirect FALSE))
=>
        (assert (change reject))
)

(defrule directConnectionsRule2
        (exists (currentConnectionIsDirect TRUE))
        (exists (changeConnectionIsDirect TRUE))
        (minChangeTime ?x)
        (currentTimeToDestination ?y)
        (changeTimeToDestination ?z&:(>= ?z (+ ?x ?y)))
=>
        (assert (change reject))
)

(defrule nextStationTimeRule
        (exists (bothConnectionsHasNextStation TRUE))
        (exists (sameNextStations TRUE))
        (minChangeTime ?x)
        (currentTimeToNextStation ?y)
        (changeTimeToNextStation ?z&:(>= ?z (+ ?x ?y)))
=>
        (assert (change reject))
)

(defrule followingStationsRule
        (currentConnectionDistance ?x)
```

```
        (changeMinDistance ?y&:(> ?y ?x))
=>
        (assert (change reject))
)
```

Załącznik XI: Arkusz zawierający szczegółowe dane z przeprowadzonych testów

Dokładne dane z przeprowadzonych testów można znaleźć w pliku „Wyniki testów.xls" stanowiącym załącznik do niniejszej pracy.

Załącznik XII: Oświadczenie o samodzielnym wykonaniu pracy

OŚWIADCZENIE

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przez mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

....

Data Piotr Nagrodkiewicz