AKADEMIA GÓRNICZO-HUTNICZA

Wydział Informatyki, Elektroniki i Telekomunikacji



## Rozwój algorytmów optymalizacyjnych dla środowiska do rozwiązywania problemów transportowych

**Prowadzący**:   
dr inż. Małgorzata Żabińska,   
dr inż. Jarosław Koźlak

**Zespół:**

Aksamit Dawid,

Salamon Alicja

**Spis treści**

[Rozwój algorytmów optymalizacyjnych dla środowiska do rozwiązywania problemów transportowych 1](#_Toc357771293)

[1. Sformułowanie zadania projektowego 4](#_Toc357771294)

[1.1 Dziedzina problemu 4](#_Toc357771295)

[1.2 Zadanie projektowe 4](#_Toc357771296)

[2. Harmonogram 4](#_Toc357771297)

[3. Opis wymagań 5](#_Toc357771298)

[3.1 Analiza wymagań funkcjonalnych 5](#_Toc357771299)

[3.2 Wymagania niefunkcjonalne 5](#_Toc357771300)

[4. Analiza istniejącego systemu 5](#_Toc357771301)

[4.1 Lokalizacja kodu algorytmu 6](#_Toc357771302)

[4.1.1 Pakiety 6](#_Toc357771303)

[4.1.2 Klasy 6](#_Toc357771304)

[4.2 Opis obecnego algorytmu 6](#_Toc357771305)

[4.2.1 Dane wejściowe 7](#_Toc357771306)

[4.2.2 Rezultat 8](#_Toc357771307)

[4.3 Konfiguracja algorytmu 8](#_Toc357771308)

[5. Analiza zastosowanego algorytmu 9](#_Toc357771309)

[5.1 Adaptive Large Neighborhood Search 9](#_Toc357771310)

[5.1.1 Metody wyjmowania zleceń ALNS: 10](#_Toc357771311)

[5.1.2 Metody wstawiania zleceń ALNS: 10](#_Toc357771312)

[5.1.3 Kombinacja heurystyk 11](#_Toc357771313)

[5.2 Metoda ruletki 11](#_Toc357771314)

[5.3 Szczegóły metody Shawna wyjmowania zleceń 11](#_Toc357771315)

[5.3.1 Cel 11](#_Toc357771316)

[5.3.2 Znaczenie 12](#_Toc357771317)

[5.3.3 Wzór 12](#_Toc357771318)

[6. Model rozwiązania 13](#_Toc357771319)

[6.1 Dodanie Algorytmu 13](#_Toc357771320)

[6.2 Klasa implementująca algorytm STLike 14](#_Toc357771321)

[14](#_Toc357771322)

[6.3 Operacje na zleceniach 14](#_Toc357771323)

[6.3.1 Wyciąganie zleceń 15](#_Toc357771324)

[6.3.2 Wkładanie zleceń 16](#_Toc357771325)

[6.3.3 Koło ruletki 17](#_Toc357771326)

[6.4 Diagram sekwencji 18](#_Toc357771327)

[7. Uruchamianie 18](#_Toc357771328)

[7.1 Pliki konfiguracyjne 18](#_Toc357771329)

[8. Metoda wyjmowania Shawna jako miara 19](#_Toc357771330)

[9. Testy 20](#_Toc357771331)

[10. Podsumowanie 22](#_Toc357771332)

[10.1 Weryfikacja poprawności 22](#_Toc357771333)

[10.2 Uwagi i wnioski końcowe 22](#_Toc357771334)

1. Sformułowanie zadania projektowego
   1. Dziedzina problemu

Problem PDPTW (Pickup and Delivery Problem with Time Windows) jest uogólnieniem problemu VRPTW (Visual Routing Problem with Time Windows), który jest z kolei uogólnieniem problemu komiwojażera. Oznacza to, że jest to problem NP-zupełny.

Rozwiązanie problemu PDPTW polega na znalezieniu zbioru tras przejazdu dla floty przejazdów w celu wykonania zestawu zleceń transportowych. Każdy pojazd ma określoną ładowność. Każde zlecenie jest zdefiniowane przez punkty załadunku i wyładunku, wymagana ładowność oraz przez czasy odbioru i dostarczenia zleceń. Dodatkowo zdefiniowany jest punkt początkowy i końcowy dla każdej z tras.

W zależności od oczekiwanego efektu końcowego możemy wyróżnić kilka celów poszukiwania rozwiązania. Istnieje kilka kryteriów według których możemy stwierdzić, że rozwiązanie jest optymalne.

Minimalizacja liczby pojazdów oznacza, że rozwiązaniem jest to, w którym liczba pojazdów będzie najmniejsza. Minimalizacja czasu to poszukiwanie rozwiązania, w którym wszystkie zlecenia zostaną wykonane w najkrótszym czasie. Minimalizacja dystansu oznacza poszukiwanie możliwie najkrótszych tras dla pojazdów. Minimalizacja czasu oczekiwania, oznacza poszukiwanie rozwiązania, w którym czas oczekiwania na realizację zleceń jest najmniejszy.

* 1. Zadanie projektowe

Nasze zadanie polegało na ewolucji systemu Dispatch Rider służącego do rozwiązywania dynamicznego problemu transportowego. Zakres naszych zmian obejmuje optymalizację algorytmu przydziału zleceń. Naszym celem jest uzyskanie lepszych wyników pracy systemu, tj. bardziej optymalny przydział zleceń (wyższy współczynnik funkcji celu).

W tym celu stworzony został nowy algorytm oparty o Adaptive Large Neighborhood Search, który jest najlepszym znanym algorytmem do rozwiązywania problemu PDPTW (dla przypadków statycznych).

1. Harmonogram

|  |  |
| --- | --- |
| 8 IV 2013 | Wstępna analiza projektu |
| 15 IV 2013 | Analiza:   * Rozbudowana analiza źródła – dokumentacji i kodu * opis proponowanego algorytmu |
| 22 IV 2013 | Podsumowanie analizy – prezentacja |
| 6 V 2013 | Projektowanie rozwiązania |
| 13 V 2013 | Implementacja   * podstawowy algorytm * możliwości ustawienia wyboru algorytmu z poziomu konfiguracji |
| 20 V 2013 | Implementacja - przygotowanie szkieletu dla nowego algorytmu |
| 27 V 2013 | Implementacja nowego algorytmu |
| 3 VI 2013 | * Testy końcowe * Opracowanie ostatecznej dokumentacji |
| 10 VI 2013 | Podsumowanie - prezentacja |
| 17 VI 2013 | Oddanie projektu |

1. Opis wymagań
   1. Analiza wymagań funkcjonalnych

|  |
| --- |
| Powstanie nowego algorytmu wymian zleceń |
| Nowo powstały algorytm ma spełniać te same zadania co klasyczny Simmulated Trading, ale ma opierać się na zupełnie innym pomyśle. Nowy algorytm jest implementacją najlepszego znanego sposobu rozwiązywania problemów statycznych PDPTW – Adaptive Large Neighbourhood Search |

|  |
| --- |
| Konfigurowanie użycia algorytmu z poziomu pliku konfiguracyjnego |
| Użycie nowego algorytmu nie może być obligatoryjne. Zarówno nowy, jak i klasyczny algorytm mogą być ustawiane w głównym pliku konfiguracyjnym. Również parametry pliku mają być konfigurowalne. |

|  |
| --- |
| Uzyskanie rezultatów lepszych od klasycznego algorytmu Simmulated Trading pod względem liczby pojazdów i total\_distance |
| Jako że algorytm jest implementacją najlepszego znanego sposobu rozwiązywania problemów statycznych, oczekiwane jest, iż wyniki otrzymane dzięki działaniu systemu Dispatch Rider będą bardziej optymalne.  Szczególnie istotne są parametry total\_distance oraz ilość pojazdów, dzięki którym można stwierdzić, czy faktycznie rezultaty są bardziej optymalne. Oczekiwane rezulataty powinny dawać mniejsze wartości tych parametrów. |

* 1. Wymagania niefunkcjonalne

|  |
| --- |
| Spójność |
| Algorytm ma być napisany spójnie z resztą istniejącego kodu. Powinien używać odpowiedniej, stworzonej wcześnie struktury pakietów. |

|  |
| --- |
| Kompletność i poprawność |
| Algorytm powinien dawać zawsze wyniki poprawne, tzn. odpowiednie do dalszego przetwarzania w innych częściach systemu. |

|  |
| --- |
| Wydajność czasowa i pamięciowa |
| Oczywistym jest, że nowy, bardziej skomplikowany algorytm będzie działał wolniej oraz wykorzystywał więcej pamięci. Powinien jednak uruchamiać się w standardowo skonfigurowanym środowisku Eclipse. |

1. Analiza istniejącego systemu

Przez podjęciem prac programistycznych konieczna była dogłębna i obszerna analiza systemu Dispatch Rider, aby określić które jego części są istotnie z punktu wprowadzanych przez nas zmian i jak można skonfigurować system, by używał zaimplementowanych nowych funkcjonalności.

* 1. Lokalizacja kodu algorytmu
     1. Pakiety

Sama metoda complexSimmulatedTrading znajduje się w Pakiecie src.algorithm.simmulatedTrading, jednakże jest więcej pakietów które nas interesują

Pakiet measure – odpowiedzialny za miary

Pakiet algorithm - Pakiet w którym znajduje się wiekszość logiki Symulatora.

Pakiety simmulatedTrading oraz STLike – pakiety odpowiedzialne za Simmulated trading, znajdujące się w pakiecie algorithm.

* + 1. Klasy

public abstract class Schedule implements Serializable – klasa kalendarza. Posiada wiele metod używanych podczas simmulatedTrading.

public class SimmulatedTrading – klasa w której znajdują się wszystkie wersje simmulatedTrading. (fullSimmulatedTrading, complexSimmulatedTrading)

* 1. Opis obecnego algorytmu

Nasza praca skupia się głównie na algorytmie wymian zleceń fullSimulatedTraiding. Po dodaniu zlecenia holon inicjuje procedurę SimulatedTraiding. Wybiera swoje najgorsze zlecenie i wysyła je do licytacji. Teraz zlecenie to jest traktowane jak każde inne zlecenie. Każdy holon wysyła swoją ofertę (koszt dodania zlecenia). Wybierany jest holon, który wysyła najlepszą ofertę. W przypadku, gdy jest to inny holon (nie ten który wysłał to zlecenie do licytacji), to teraz on inicjuje SimulatedTraiding. Gdy wysłne na licytację zlecenie wraca do nadawcy, to wtedy wysyłane jest kolejne „najgorsze zlecenie”. Procedura kończy się, nie ma już zleceń do wysłania (wartość zmiennej STDepth jest większa od ilości zleceń w kalendarzu).

* + 1. Pseudokod

/\* Przekazywane parametry:

\* holons – mapa której kluczami są identyfikatory holonów, a \* wartościami ich kalendarze (Schedule). Reprezencja holonów w

\* symulacji holon – jest to klucz z mapy *holons* aktualnego holonu \* (inicjującego ST) STDepth – określa które najgorsze zlecenie ma \* być brane pod uwagę\* algorithm – algorytm przydziału zleceń w \* obrębie holonu \* commissionsId – zbiór identyfikatorów zleceń, \* które były już wysyłane na aukcję w ramach ST (służy zapobieżeniu \* zapętlenia algorytmu) time – określa czy najgorsze zlecenie ma

\* być wybierane po czasie bezczynności (time=false), czy ogólnym \* czasie dostarczenia wszystkich zleceń (time=true)

\* timestamp – obecny znacznik czasowy symulacji

\*/

Map<AID, Schedule> fullSimmulatedTrading(holons, holon, STDepth, depot, algorithm, commissionsId, time, timestamp)

begin

schedule = holons.get(holon)

backup = utwórz kopie bieżącego kalendarza (schedule)

worstCommission = pobierz najgorsze zlecenie z bieżącego kalendarza (uwzględniając zmienną time, oraz STDepth)

/\* worstCommission jest null’em, gdy STDepth jest za duże (większe od ilości zleceń   
 w kalendarzu)\*/

if(worstCommission==null) then

begin

holons.put(holon,backup);

return holons;

end

if(commissionsId.contains(worstCommission.getID()) then

begin

return holons;

end

commissionsId.add(worstCommission.getID());

for każdy holon (h) do

begin

schedule = pobierz kalendarz holonu h

if worsCommission może być dodane do kalendarza schedule then

begin

extraDistance=(dystans po dodaniu zlecenia) – (dystans przed dodaniem)

if(bestCost>getRatio(extraDistance,worstCommission) then

begin

bestCost=getRatio(extraDistance,worstCommission);

bestHolon=h

bestSchedule=schedule

end

added=true;

end

end

if(added==false) then

begin

holons.put(holon,backup);

return holons;

end

holons.put(bestHolon,bestSchedule)

if(bestHoilon==holon) return fullSimmulatedTrading(holons, holon, STDepth+1,   
 depot, algorithm, commissionsId, time)

else return fullSimmulatedTrading(holons, bestHolon, 1, depot, algorithm, commissionsId, time)

end

* + 1. Rezultat

Rezultatem wykonania algorytmu jest nowa mapa holonów w której zaktualizowane są ich kalendarze lub stara mapa holonów jeśli nie powiodło się wymiana zleceń.

* 1. Konfiguracja algorytmu

Elementy te związane są z wprowadzaniem algorytmów optymalizacji działających podobnie, jak obecne SimmulatedTrading.

|  |  |
| --- | --- |
| exchangeAlgorithmAfterComAdd | |
| Określa parametry dla algorytmów STLike. Jeśli jest używany, to wszystkie parametry  dotyczące ST są ignorowane. Metoda wołana jest wtedy, co poprzednio  fullSimmulatedTrading. | |
| name | Nazwa algorytmu(klasy). |
| param | Za pomocą par name i value podajemy odpowiednie argumenty o odpowiednich wartościach dla algorytmu. |

|  |  |
| --- | --- |
| exchangeAlgorithmWhenCantAddCom | |
| Określa parametry dla algorytmów STLike. Jeśli jest używany, to wszystkie parametry  dotyczące ST są ignorowane. Metoda ta wołana jest wtedy, co poprzednio  complexSimmulatedTrading. | |
| name | Nazwa algorytmu(klasy). |
| param | Za pomocą par name i value podajemy odpowiednie argumenty o odpowiednich wartościach dla algorytmu. |

Istotne są też główne parametry symulacji:

|  |  |
| --- | --- |
| Commitions | |
| Konfiguruje główne parametry symulacji. | |
| dynamic | Domyślnie „false”. Określa, czy problem jest statyczny, czy dynamiczny. |
| simmulatedTrading | Domyślnie „1”. Określa, ile razy ma być uruchamiane simulatedTrading. |
| STDepth | Domyślnie „1”. Stopień zagłębienia w algorytmie complexSimmulatedTrading. |
| firstComplexSTResultOnly | Domyślnie „true”. Określa tryb działania algorytmu complexSimmulatedTrading. Czy ma być wybierana pierwsza  znaleziona konfiguracja zleceń, czy też konfiguracja optymalna.. |
| STTimeGap | Domyślnie „1”. Ustawienie częstości uruchamiania SimulatedTrading w zależności od ilości znaczników czasowych. |

1. Analiza zastosowanego algorytmu

Przed implementacją poddaliśmy algorymt ALNS dokładnej analizie.

* 1. Adaptive Large Neighborhood Search

Adaptive Large Neighborhood Search jest algorytmem rozszerzającym Large Neighborhood Search,  
który opiera się na wprowadzeniu optymalizacji poprzez wprowadzenie bardzo dużych ruchów (zmian) w aktualnie przyjętym rozwiązaniu. Zmiany takie powodują zmianę przydziału nawet 30-40% zleceń. Podczas gdy LNS najczęściej jest oparty na stosowaniu skomplikowanych i ciężkich metod wstawiania i wyjmowania zleceń, ALNS używa prostych metod do tego celu. Co czyni ten algorytm dobrym, jest fakt, że tych metod jest kilka i są używane zależnie od statystyk zebranych podczas pracy.

* + 1. Metody wyjmowania zleceń ALNS:
    2. Metody wstawiania zleceń ALNS:
    3. Kombinacja heurystyk

W algorytmie używane są wszystkie opisane heurystyki – zarówno te do wstawiania, jak i do usuwania zleceń. Każda z nich ma nadaną wagę. Wybieranie heurystyk opiera się na zasadzie koła ruletki. Prawdopodobieństwo wyboru wynosi , gdzie w to waga odpowiedniej heurystyki. Wagi są nadawane automatycznie bazując na statystykach zebranych podczas wcześniejszych iteracji.

* 1. Metoda ruletki

Wprowadzamy zmianę wyboru najlepszego rozwiązania. Zastosujemy ruletkę która będzie wybierać rozwiązanie proporcjonalnie do jakości wyniku rozwiązania – czyli czym lepszy wynik ma rozwiązanie tym większa szansa, że zostanie wybrane.

* 1. Szczegóły metody Shawna wyjmowania zleceń
     1. Cel

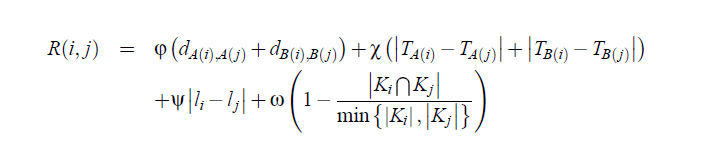
Wyciąganie metod które są podobne do siebie pod każdym z tych względów. Współczynniki służą nam do dostosowania się do sytuacji np. mając pojazdy o dużej ładowności i małe ilości towaru, który musimy w dużo miejsc rozwieść, naturalne rozwiążemy skupiając się na lokalizacji tych miejsc oraz czasie odbioru/dowozu towaru. Jednakże w większości sytuacji interesują nas tylko takie zlecenia, które są podobne pod każdym względem.

* Zlecenia, które odbywają się w podobnym czasie lecz w kompletnie innym miejscu lub dotyczące kompletnie różnych ilości towaru, w większości przypadków wrócą w to samo miejsce.
* Zlecenia, które są podobne pod każdym względem dają znacznie większe szanse na pojawienie się możliwych zamian.
  + 1. Znaczenie

Metoda polega na wyjmowaniu zleceń które są podobne do siebie. Obliczanie podobieństwa zleceń opiera się na obliczeniu czterech składowych:

* składowa odległości
* składowa czasu
* składowa obciążenia
* składowa odpowiadająca zależna od ilości pojazdów które mogą wykonywać dane zlecenia
  + 1. Wzór

Wartość miary przedstawia wzór:



Współczynniki oznaczone greckimi literami służą do ustalenia wag poszczególnych składowych.

Wzór można wytłumaczyć w następujący sposób

podobieństwo (zlecenie i, zlecenie j) =

współczynnik odległości \* (odległość między położeniem miejsc odebrania towaru obu zleceń + odległość między położeniem miejsc dowozu towaru obu zleceń)   
+

współczynnik czasu \* ( wartość bezwględna z różnicy czasów odbioru towaru obu zleceń + wartość bezwględna z różnicy czasów dowozu towaru obu zleceń)  
 +  
współczynnik pojazdów \* ( 1 - (ilość pojazdów która jest w stanie obsłużyć zarówno zlecenie i jak i zlecenie j) / ( min ( ilość pojazdów które są w stanie obsłużyc zlecenie i, ilość pojazdów które są w stanie obsłużyc zlecenie j)) )

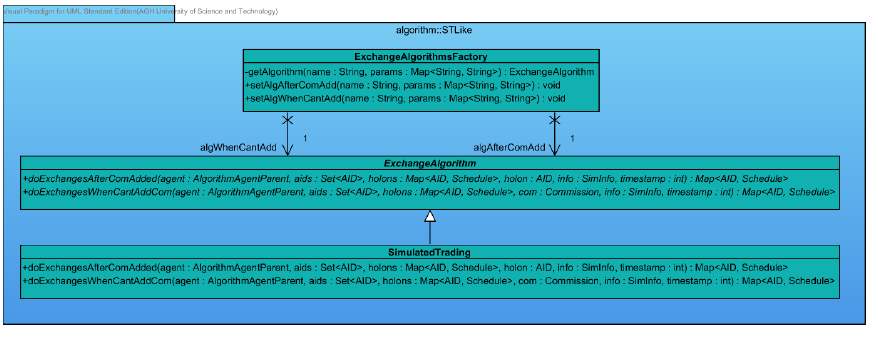
Niektóre zmienne naturalnie są przeskalowane by dawać rozsądny wynik.

Wynik tej funkcji celu nie jest ostatecznym kryterium, ponieważ dołączony parametr 'p' który będzie decydował o losowości tej metody.

1. Model rozwiązania
   1. Dodanie Algorytmu

System umożliwia proste dodawanie algorytmów działających podobnie do SimmulatedTrading.

Przedstawia to poniższy diagram:



Etapy

1. Stworzyć i dodać klasę do pakietu STLike
2. Klasa musi rozszerzać ExchangeAlgorithm a co za tym idzie musimy zaimplementować dwie metody:

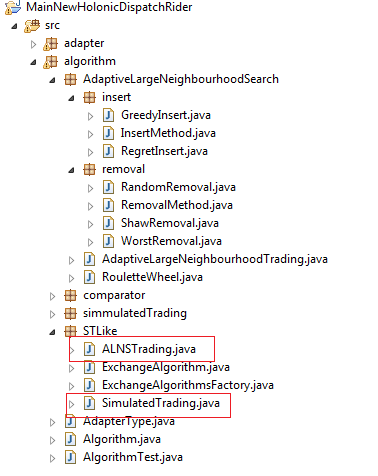
* doExchangesAfterComAdded – metoda ta jest wołana wtedy, co poprzednio algorytm fullSimmulatedTrading
* doExchangesWhenCantAddCom – metoda wołana wtedy co poprzednio algorytm complexSimmulatedTrading

1. Konfiguracja

* exchangeAlgorithmAfterComAdd
* exchangeAlgorithmWhenCantAddCom

Trzeba te parametry ustawić tak, by wskazywały na nasz nowo dodany Algorytm.

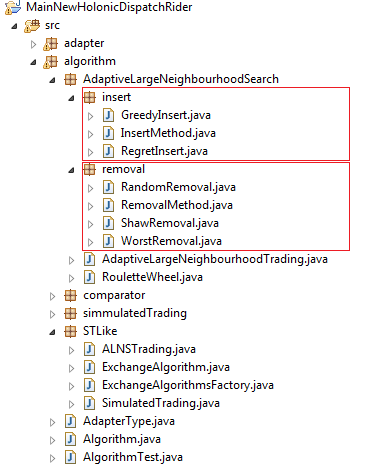
* 1. Klasa implementująca algorytm STLike



Zaznaczona klasa ALNSTrading posiada wywołania metod fullSimulatedTrading() oraz metodę zastępczą dla ComplexSimulatedTrading, czyli AdaptiveLargeNeighbourhoodTrading.

Wybór tej klasy w konfiguracji sprawia, że zamiast domyślnie używanej klasy SimulatedTrading będzie stosowany nowo implementowany algorytm.

Powyższy pakiet AdaptiveLargeNeighbourhoodSearch zawiera zarówno implementację nowej metody wymian zleceń oraz klasy i interfejsy potrzebne w działaniu algorytmu.

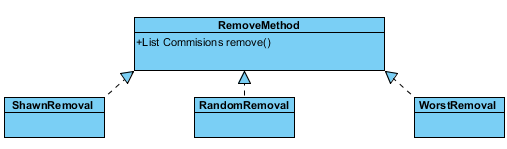


* 1. Operacje na zleceniach

Pakiet AdaptiveLargeNeighbourhoodSearch.insert zawiera interfejs InsertMethod oraz implementujące go klasy GreedyInsert oraz RegretInsert odpowiedzialne za wstawianie zleceń w algorytmie AdaptiveLargeNeighbourhoodTrading.

Pakiet AdaptiveLargeNeighbourhoodSearch.removal zawiera interfejs RemovalMethod oraz implementujące go klasy RandomRemoval, ShawnRemoval oraz WorstRemoval odpowiedzialne za wyciaganie zleceń w algorytmie AdaptiveLargeNeighbourhoodTrading.

* + 1. Wyciąganie zleceń

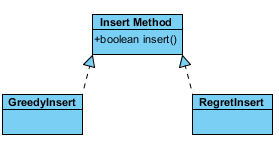


|  |
| --- |
| Losowa metoda wyjmowania |
| Jest to losowe wyjmowanie zlecenia z kalendarza. |

|  |
| --- |
| Wyjmowanie Shawna |
| Jest to wyjmowanie oparte na wyszukiwaniu zleceń które są w pewien sposób podobne do siebie. Tego rodzaju zlecenia powinny powinny być łatwe do wymiany. |

|  |
| --- |
| Wyjmowanie najgorszego zlecenia |
| Jest to metoda oparta na wyjmowaniu zlecenia które najbardziej obciąża dane rozwiązanie. Metoda ta ma na celu przeniesienie zleceń, które są w nieodpowiednim miejscu |

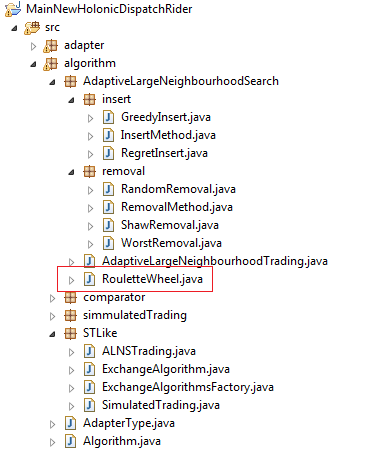
* + 1. Wkładanie zleceń



|  |
| --- |
| Zachłanne wstawianie |
| Wstawianie po kolei zleceń w dane miejsce na danej drodze, gdzie koszt tego zlecenia będzie najniższy. Jeśli nie uda się wstawić to zostawiamy zlecenie i zostanie wstawione w innej ścieżce, a jeśli się uda to nie sprawdzamy czy bardziej się opłaca wstawić w innych ścieżkach. Ma to na celu utrzymanie dużej szybkości algorytmu. Algorytm ten ma problem ze zleceniami które ciężko wstawić i które są zostawiane do ostatnich iteracji ze wstawieniem i mogą być bardzo nieopłacalne. Temu problemowi sprzeciwia się kolejna metoda wstawiania. |

|  |
| --- |
| Metoda wstawania "Regret" |
| jest to wstawianie do pewnego stopnia chciwe, lecz w tej metodzie patrzymy w przód i sprawdzamy, jak dużo stracimy jeśli nie wstawimy w danego zlecenia w jego optymalne miejsce. Jeśli przyjmiemy, że funkcją celu w zachłannym wstawianiu była minimalizacja kosztu, w tej metodzie jest nią maksymalizacja "Regret" czyli dodatkowego kosztu który wynika z NIE wstawienia metody w jej optymalne miejsce. W tym algorytmie możemy podać, ile "kolejnych najlepszych" miejsc wstawienia oraz ich kosztów sprawdzamy. |

* + 1. Koło ruletki

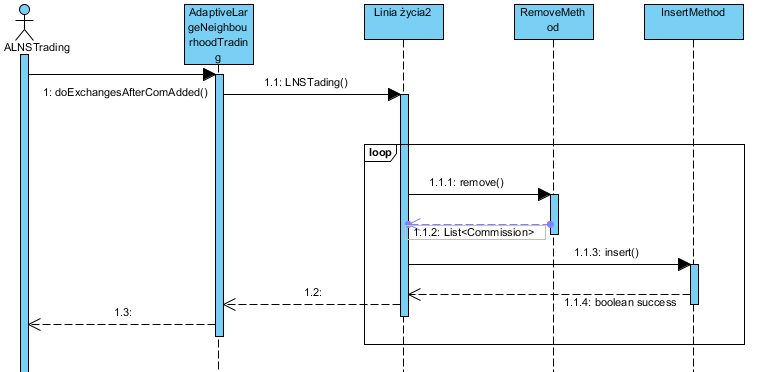


Klasa RouletteWheel zawiera metody, które na podstawie zebranych statystyk wyznaczają konkretne klasy wstawiania i wyciągania zleceń do użycia w algorytmie.

Metoda RemovalMethod getRemovalMethod() zwraca odpowiednią dla konkretnego wywołania metodę wyciągania zleceń.

Metoda InsertMethod getInsertMethod() zwraca dla konkretnego wywołania metodę wstawiania zleceń.

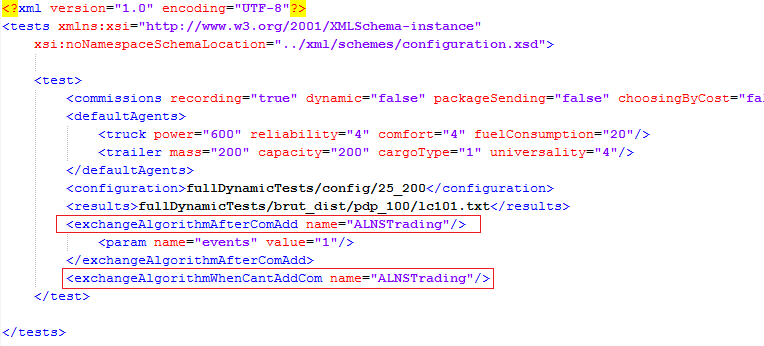
* 1. Diagram sekwencji

****

1. Uruchamianie
   1. Pliki konfiguracyjne

Aby wykorzystać implementowany nowy algorytm należy uruchomić system z odpowiednio zmodyfikowanymi plikami konfiguracyjnymi.

W przykładowym pliku konfiguracyjnym dodane parametry exchangeAlgorithmAfterComAdd oraz exchangeAlgorithmWhenCantAddCom pozwalają na jego uruchamianie.



Dodatkowo każdy algorytm może mieć ustalane parametry za pomocą par name i value.



1. Metoda wyjmowania Shawna jako miara

Istnieje kilka problemów ze stosowaniem tej metody jako miary - wynik nie ma sensownego sposobu odczytania. Czemu tak jest? Otóż metoda ta normalnie jest stosowana do wylosowania pojedynczego zlecenia a później szukania zleceń mu podobnych. Jako, że miary działają tak, że przypisują pewien wynik (miarę) do każdego holonu, musimy przypisać każdemu holonowi jakąś miarę związaną z podobieństwem wszystkich zleceń ( zakładamy tutaj, że miara ma opisywać cały holon a nie, np. wylosowane jedno zlecenie). Powstałe problemy:

* nie liczymy porównania jednego zlecenia do wszystkich, przyjmując powyższe założenie możemy rozwiązać ten problem na 2 sposoby. Jedna możliwość to porównanie wszystkich zleceń - każdy z każdym, a druga to porównanie zleceń z jednego holonu. W pierwszym przypadku mamy do czynienia ze sporym nakładem obliczeniowym podczas gdy obie z tych sytuacji cierpią z kolejnego większego problemu:
* nie ma żadnego rozsądnego sposobu odczytania wyników takie miary. Musimy stworzyć jedną liczbę z ogromnej ilości liczb, które same w sobie mają skomplikowaną wzór. Z czym więc kończymy? Dla przykładu, możemy skończyć z holonem który ma 10 zleceń oraz z wartością funkcji podobieństwa dla każdej pary(45 unikalnych wyników, a możliwe setki lub tysiące wyników jeśli spróbujemy porównać ze zleceniami z innych holonów, czyli tak jak jest gdy używa się tej metody w zamierzony sposób). Teraz mając te 45-kilkaset wyników musimy zrobić jedną liczbę która opisze nam jeden pojazd. Możemy zrobić średnią lub sumę lub użyć innej funkcji lecz zawsze zostaniemy z liczbą która nam nie daje żadnej 'miary'. Wyjmowanie to działa, ponieważ zlecenia o podobnych składowych są zleceniami które można zamienić, jednakże jeśli przeniesiemy to na pojazdy to porównywanie liczb je opisującej nie posiada dla nas żadnej wartości. Nie jesteśmy w stanie stwierdzić po tej liczbie czy dane 2 samochody mają cokolwiek podobnego, ponieważ ten wynik jest mieszanką tak dużej ilości innych wyników że może to oznaczać mnóstwo różnych rzeczy i my nie jesteśmy w stanie stwierdzić co dany wynik kiedy oznacza. Co więcej, dwa zlecenia porównane z wszystkimi innymi zleceniami nawet jeśli po funkcji celu będą wyglądać jak podobne to w większości przypadków będą zlecenia które nie mają ze sobą nic wspólnego a co za tym idzie, wynik będzie nie miarodajny.

1. Testy

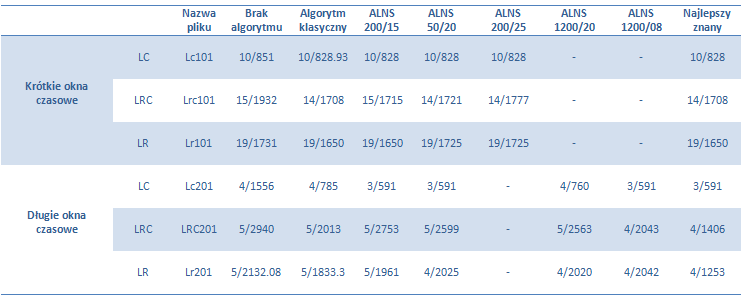
Pliki testowe dla problemy PDPTW można podzielić według dwóch klasyfikacji

1. Ze względu na długość okien czasowych (długie i krótkie)
2. Ze względu na ilość i skupienie zleceń (LR, LRC, LR)

Testy zostały przeprowadzone dla wszystkich sześciu kategorii dla 3 konfiguracji:

1. Bez dodanego algorytmu STlike
2. Z klasyczną wersją algorytmu
3. Z algorytmem ALNS

Oraz dodane są dane o najlepszych znalezionych wynikach ze strony <http://www.sintef.no/Projectweb/TOP/PDPTW/Li--Lim-benchmark/>

Wartości w tabeli to ilość pojazdów/ total distance. Parametry przy algorytmie ALNS oznaczają ilość iteracji/% zleceń wyciąganych

1. Podsumowanie

Zaimplementowany przez nas algorytm bardzo dobrze spisuje się w testach. W tych z długimi oknami czasowymi znajduje rozwiązania lepsze nie tylko pod względem dystansu, ale także ilości pojazdów. Niestety są to wyniki wciąż dalekie od najlepszych znanych. Może to wynikać z faktu, że testy w warunkach lokalnych nie mogły być uruchomione na wystarczająco dużej ilości iteracji, a może też z nieodpowiedniego wyboru wielkości zmian w rozwiązaniach.

Jednakże są to wciąż bardzo dobre wyniki –uzyskana została poprawa dla trudnych testów i warto w nich stosować ten algorytm. Do prostych testów, szczególnie krótkich okien czasowych można wciąż używać starego algorytmu, jako że osiąga tak samo dobre wyniki mniejszym nakładem czasowym i obliczeniowym. Nasza implementacja została bardzo ukierunkowana przez resztę systemu i wiele rozwiązań przez nas zastosowanych dało by się zmienić w innych okolicznościach.