Akademia Górniczo-Hutnicza

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

Katedra Informatyki

Planowanie działań i identyfikacja sytuacji w wieloagentowym systemie transportowym przy pomocy środowiska R

Wersja 1.1 z dnia 23.01.2013 r.

Kierunek, rok studiów:
Informatyka, I rok, st 2

Przedmiot:
Systemy zdecentralizowane i agentowe

Pod kierownictwem:
Dr inż. Jarosław Koźlak

Rok Akademicki: ***(2012/2013)***
Semestr: ***zimowy***

Zespół autorski:
Jarosław Dąbrowski
Sebastian Feduniak

Kraków 2012 r.

Spis treści

1. Wstęp

Projekt ma na celu dodanie nowej funkcjonalności do już istniejącego systemu pod nazwą „Dispatch Rider". Dodatkowym założeniem jest wykorzystanie do tego celu platformy R.

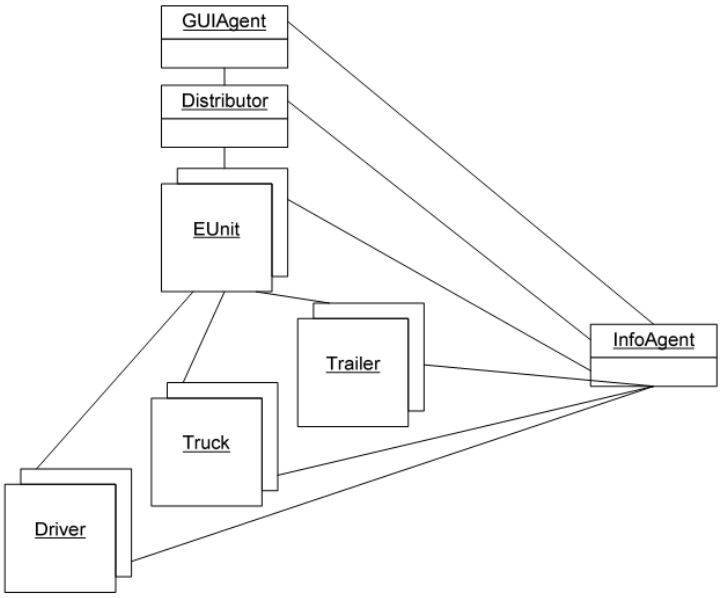
1.1 Wprowadzenie do problemu PDPTW.

PDPTW (Pickup and Delivery Problem with Windows) to problem transportowy, występujący w dwóch podstawowych wariantach: statystycznym – zestaw zleceń jest znany przed rozpoczęciem obliczeń oraz dynamicznym – zlecenia napływają w czasie działania systemu. Zadanie polega na znalezieniu zbioru optymalnych tras przejazdu dla floty pojazdów transportowych, w celu wykonania grupy zleceń transportowych. Przed przejściem do rozwiązywania problemu określa się pewne ograniczenia, takie jak : pierwszeństwa, par, ładowności, zasobów. Ograniczenia są jednocześnie wstępną konfiguracją systemu. Przed przystąpieniem do symulacji określa się pewną funkcję celu służącą do zminimalizowania określonych wartości np. łącznego czasu wykonywania grupy zleceń. Można również minimalizować kilka wartości jednocześnie przy założeniu, że określimy dla nich odpowiednie wagi. Systemy, które rozwiązują taki problem, mogą być stosowane w przemyśle transportowym.

1.2 Istniejący system.

Naukowcy z AGH już od wielu lat prowadzą badania nad rozwiązaniem powyższego problemu. Wynikiem tych badań jest między innymi wieloagentowy system „Dispatch Rider". Oprócz rozwiązywania problemu klasycznego został wzbogacony o uwzględnianie elementów problemu dynamicznego: pojazdy poruszają się po grafie definującym sieć transportową, zlecenia mogą przybywać w trakcie symulacji, może zmieniać się stan floty transportowej, mogą występować sytuacje kryzysowe. Koncepcja rozwiązania wykorzystuje jednostki holonów do współpracy w realizacji grupy zleceń. System bardzo rozwinięty: oprócz symulacji umożliwia wizualizację oraz gromadzi informacje które mogą być wykorzystane w kolejnych testach (uczenie maszynowe).

W celu rozwiązania tego problemu została stworzona następująca struktura agentów :



- Distributor – przydzielanie zleceń
- Eunit – kompletuje skład zespołu (holon), zarządza trasą przejazdu i obliczna jej koszt
- Driver, Truck, Trailer – agenci transportowi
- InfoAgent – tworzenie agentów, przechowywanie informacji na ich temat
- GUIAgent – generowanie interfejsu użytkownika

1.3 JADE jako platforma agentowa do rozwiązywania problemów typu PDPTW

System wykorzystuje platformę JADE, która jest napisanym w języku Java framework'iem wspomagającym budowę systemów wieloagentowych. Framework ten pozwala konstruować oraz administrować agentów zgodnych z Foundation for Intelligent Physical Agents. Dodatkowo posiada wiele narzędzi graficznych, które wspierają debugowanie oraz wdrażanie aplikacji agentowych Wspiera ona wiele obszarów adresowanych w trakcie realizacji multiagentowego systemu:

- Komunikacja agentów przez wymianę wiadomości – model asynchroniczny, agenci posiadają skrzynki odbiorcze
- Zgodność wiadomości ze standardem ACL – ułatwia selekcję wiadomości
- Rozproszenie systemu – system można rozporszyć na wiele systemów tak aby wykorzystać ich moc obliczeniową do rozwiązania problemu
- Interfejs użytkownika – ułatwia kontrolę nad przebiegem symulacji

1.4 Platforma R

Rozszerzenie projektu o zaprojaktowany przez nas model uczenia maszynowego, w dużym stopniu będzie korzystało z platformy R. R jest jednocześnie językiem programowania jak również środowiskiem do obliczeń statystycznych. Posiada on wiele matematycznych funkcji, w tym operacje na macierzach. Szczególnie przyda nam się wbudowany w język silnik do analizy danych, z naszego punktu widzenia istotna jest funkcja clusteringu zbioru danych. Dane w R można bardzo łatwo wizualizować. Co dla nas również ważne R udostępnia interfejs dzięki, któremu można się z

nim połączyć z poziomu Javy, jest to dla nas pomocne szczególnie z tego względu, iż „Dispatch Rider” jest zaimplementowany w Javie. Dodatkowo należy wspomnieć, że kod jest otwarty, można go używać i modyfikować za darmo.

1.5 Uczenie maszynowe

Przed przejściem do opisu tego jak wyborażamy sobie nasze rozwiązanie, należy wspomnieć o tym czym jest uczenie maszynowe. Należy to uczynić z tego względu iż uczenie maszynowe będzie przez nas wykorzystywane w pewnym stopniu jak również zostało one już użyte w systemie „Dispatch Rider”.
Ogólnie rzecz biorąc uczenie maszynowe to młoda i szybko rozwijająca się dziedzina wchodząca w skład nauk zajmujących się sztuczną inteligencją.
Można ją rozumieć jako metodę konkretyzacji algorytmu, czyli dobór parametrów nazywanych wiedzą lub umiejętnością. Wyróżnia się wiele metod pozyskiwania takiej wiedzy.
W ogólności, uczenie maszynowe stosuje się w celu zwiększenia:

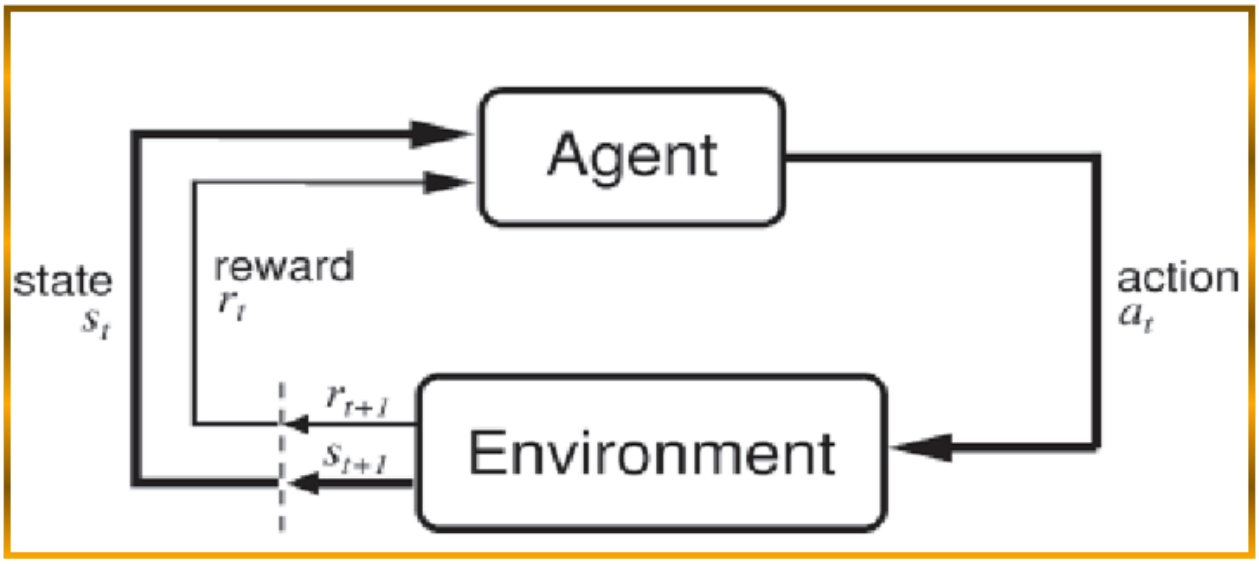
- Efektywności
- Wydajności
- Bezawaryjności
- Redukcji kosztów

W rozwijanym systemie, uczenie maszynowe umożliwa automatyczny dobór parametrów symulacji w zależności od jej stanu.

1.6 Q-Learning – zastosowana metoda uczenia maszynowego

Do automatycznego doboru parametrów symulacji w zależności od jej stanu użyto Q-Learningu, który polega na sekwencyjnym uczeniu systemu na podstawie odpowiednio przygotowanej tabeli. Q-Learning zakłada, że mamy wyróżniony zbiór stanów S w jakich może znajdować się system oraz zbiór akcji do wykonania A. Używamy do tego reprezentacji tabelowej, w której stany reprezentują wieksze, a akcje kolumny. Wykonania akcji może spowodować przejście między stanami. Po wykonaniu dowolnej akcji w stanie S na podstawie istniejących wyników obliczania jest nagroda za pomocą funkcji nagrody. Wartość komórki [s,a] jest aktualizowana na podstawie nowego stanu oraz wartości nagrody.
Jako, że system pozwala na wyliczanie różnych miar podczas działania symulacji oraz możliwa jest dynamiczna zmiana konfiguracji systemu w czasie jego działania, to zdecydowano się na użycie Q-learningu.
Celem użycia Q-Learningu w tym wypadku było zmniejszenie czasu obliczeń oraz poprawienie otrzymanego rozwiązania.
Użyto Q-learningu do wyboru optymalnych w danej chwili algorytmów przydziału zleceń.

Poniższy rysunek ładnie obrazuje jak wygląda mechanizm Q-Learningu :



W przypadku „Dispatch Rider” algorytm wygląda w następujący sposób :

- na początku symulacji wczytywany jest plik z opisem tabel
- dokonywana jest inicjalizacja tabeli (na podstawie zawartości poprzednio wczytanego pliku)
- po przybyciu każdego nowego zlecenia:
  - wyliczane są wartości miar i wybierany jest aktualny stan systemu (s’)
  - wybierana jest akcja dla aktualnego stanu (a’)
  - wyliczana jest nagroda (reward) po zmianie stanu s na s’
  - wartość komórki dla (s, a) jest updateowana przez nową wartość
  - s = s’, a = a’
- po zakończeniu symulacji zawartość tabeli jest zapisywana do pliku (żeby można było użyć jej w kolejnej symulacji)

1.7 Wady użytego rozwiązania

Użyte rozwiązanie mimo tego, że daje bardzo dobre rezultaty ma parę znaczących wad :

- istnieje stały oraz ograniczony zbiór stanów systemu – w konfiguracji podany jest zbiór stanów podany przez użytkownika,
- dobór stanów wymaga doskonałej znajomości zagadnienia oraz wykonania szeregu testowych symulacji jak również obserwacji wyników,
- wybór innej funkcji funkcji nagrody może wymuszać potrzebe przebudowania stanów – zmiana funkcji nagrody może spowodować, że niektóre akcje nie będą w ogóle wykonywane lub niektóre stany nie będą używane, tak czy siak należy w takiej przebudować akcje i/lub stany,
- dodanie nowej miary lub zmiana istniejącej może wymusić potrzebę przebudowania stanów

2. Wizja projektu

2.1 Cel projektu

Celem projektu jest rozwinięcie i jednocześnie poprawienie działania istniejącego systemu „Dispatch Rider". Zakładamy dodanie nowego algorytmu uczenia maszynowego, który będzie udoskonaleniem istniejącego – QLearning. Cel, do którego dążymy to automatyczna generacja stanów systemu tak, aby nie trzeba było ich wprowadzać ręcznie i zmieniać. System na podstawie historii swojego działania ma się uczyć, jakie wartości measurmentów występowały i na tej podstawie identyfikować stany systemu. Chcemy zostać przy koncepcji, w której wykorzystywana jest tablica oraz na stałe określone są akcje systemu czyli możliwe zmiany konfiguracji działania. Do rozwiązania tego problemu zakładamy użycie algorytmów klastrujących, które pomogą nam na podstawie measurmentów wyliczyć stany systemu. W ramach realizacji chcemy użyć pakietu R wraz z jego modułami, w których znajdują się rożne algorytmy clusteringu, których chcemy użyć. Na początku chcielibmyśmy użyć jednego z najpopularniejszych algorytmów klastrowania : k-means. Dodatkowo używane będą dwa algorytmy (wybór w konfiguracji) : pam oraz clara. Zakłada się wporawdzenie dwóch trybów : uczenia oraz pracy. W trybie uczenia system będzie wykonywał normalną pracę, ale dodatkowo zbierane będą measurmenty, na podstawie których na końcu będziemy mogli wyróżnić istniejące stany systemu i zapisać taką konfigurację. Tryb pracy zakłada, że system zna stany wygenerowane w trybie nauki i nowe measurementy dopasowuje do jednego z nich. Zakładamy możliwość wyboru dwóch trybów predykcji stanu. Pierwszym będzie predykcja na podstawie przynależności punktu do klastra. Druga metoda polega na budowie drzewa decyzyjnego, a następnie używania go do wyznaczenia stanu systemu. Dodatkowo, nasz system powinien być w maksymalnym stopniu konfigurowalny.

Zalety projektu :

- Automatyczna generacja stanów systemu
- Konfiguracja systemu możliwa do zmiany (plik xml)
- Użycie R
- Użycie szybkich oraz wydajnych algorytmów clusteringu
- Możliwość pracy w dwóch trybach (nauki i pracy)
- Dwa sposoby predykcji stanu systemu
- Wybór algorytmu dla k-means

Dalsze możliwości rozwoju :

- Użycie innych algorytmów clusteringu, np. klastrowanie hierarchiczne
- Możliwość generowania measurmentów, które będą brane pod uwagę przy clusteringu.

2.2 Wymagania funkcjonalne

System :

- musi na podstawie measurmentów generować stany systemu
- musi pozwalać na konfigurację akcji globalnych oraz lokalnych
- musi pozwalać na konfigurację measurmentów globalnych oraz lokalnych
- musi korzystać z dostarczonej konfiguracji w pliku XML
- musi używać R jako platformy do obliczeń
- po symulacji musi zapisywać na nowo konfigurację do pliku, w tym nowo nauczone stany oraz wypełnioną tabelę (stany na akcję)
- musi umożliwiać podanie czy stara konfiguracja ma być nadpisana
- może używać k-means jako algorytmu klastrującego
- powinien używać dwóch trybów pracy : nauki i pracy
- w trybie nauki powinien zapisywać sobie measurmenty, aby na końcu stworzyć z nich klastry – stany
- w trybie nauki powinien zapisywać sobie measurmenty, na końcu zapisać je do pliku wraz z przynależnością do stanu aby można ich było użyć trybie pracy do budowy drzewa decyzyjnego,
- w trybie pracy powinien dopasowywać measurmenty do istniejących klastrów – stanów i przy włączonym trybie eksploracji uczyć się – zmieniać komórki tabeli
- powinien mieć możliwość wyboru sposobu predykcji : predict z clue lub drzewa decyzyjne,
- powinien pozwalać na zakres ilości klastrów jaka może być generowana, np. od 2 do sqrt(N) – gdzie N to ilość pomiarów,
- powinien generować optymalną ilość klastrów (stanów systemu)

2.3 Wymagania niefunkcjonalne

System :

- powinien być w pełni konfigurowalny
- powinien działać szybko i niezawodnie
- powinien być możliwy do wyłączenia (wybór innego algorytmu uczenia maszynowego)
- włączenie nie powinno mieć wpływu na resztę systemu
- powinien dawać dobre wyniki – podobnie jak przy zastosowaniu q-learningu
- powinien działać szybko – nie dużo wolniej niż q-learning
- kod powinien być wersjonowany i przechowywany w repozytorium SVN

3. Koncepcja rozwiązania

W naszej pracy chcemy się skoncentrować na rozwiązaniu problemów istniejącego rozwiązania. Przede wszystkim chodzi nam o automatyczną generację stanów. Do generacji stanów użyjemy klastrowania, będzie ono zachodziło w trybie nauki. W trybie pracy do wyboru stanu na podstawie measurmentów użyjemy dwóch algorytmów : dopasowanie do środków klastrów (cluster predict) lub drzewa decyzyjne. Do implementacji wszystkich podanych wyżej algorytmów użyjemy platformy R.

Koncepcja zakłada podział całej pracy systemu na trzy etapy:

i) analiza skupień – pierwszy etap pracy, na podstawie przeprowadzonych symulacji i zgromadzonych pomiarów zachodzi wyróżnienie stanów systemu, w obecnej chwili ma tu zastosowanie algorytm klastrowania kmeans, możliwa jest też implementacja klastrowania hierarchicznego
ii) predykcja stanu systemu – drugi etap pracy, na podstawie zapisanych stanów oraz wartości pomiarów w danym kroku symulacji, wybierany jest aktualny stan systemu, obecnie do wyboru jest predykcja na podstawie środków klastrów oraz na podstawie drzew decyzyjnych
iii) wybór akcji do wykonania – trzeci etap pracy, na podstawie aktualnego stanu systemu wybierana jest akcja do wykonania, ma tu zastosowanie Qlearning

Warto podkreślić, że aplikacja jest tak zaprojektowana i w taki sposób implementowana, aby wybór algorytmu na danym etapie pracy był niezależny od algorytmów użytych w innych etapach. Dodatkowo, krytycznym założeniem jest aby wybór algorytmu w minimalnym stopniu wpływał na symulację, tzn powinien objawiać się tylko w momencie predykcji stanu systemu.
Od strony technicznej zostały wyróżnione:

i) faza inicjalizacji symulacji – zachodzi przed fazą wykonywania symulacji, przez co nie wpływa na nią czasowo, w tej fazie uruchamiany jest R oraz tworzone są struktury danych potrzebne do pracy systemu, w zależności od konfiguracji mogą to być np. drzewa decyzyjne
ii) faza wykonywania symulacji – właściwa symulacja, w trakcie wykonywania zachodzi predykcja stanu z wykorzystaniem struktur danych stworzonych w poprzedniej fazie
iii) faza finalizacji symulacji – zachodzi po zakończeniu symulacji, przez co nie wpływa na nią czasowo, w tej fazie zapisywane są wyniki symulacji

3.1 Klastrowanie

Przed przejściem do opisu pomysłu, chcemy zapoznać czytelnika z pojęciem: klastrowanie. Jest to metoda dokonująca grupowania elementów we względnie jednorodne klasy. Zakłada się, że elementy z danej klasy powinny być jak najbardziej podobne do siebie i jak najbardziej różne od elementów z innych klas. Podstawą grupowania jest podobieństwo pomiędzy elementami – wyrażone przy pomocy funkcji (metryki) podobieństwa. Całość polega na podziale zbioru elementów na grupy (klasy, podzbiory). Klastrowanie między innymi służy do zredukowania dużej liczby danych pierwotnych do kilku podstawowych kategorii, które mogą być traktowane jako przedmiot dalszej analizy.

Wyróżniamy trzy podstawowe metody klastrowania :

- Metody hierarchiczne – algorytm tworzy dla zbioru obiektów hierarchię klasyfikacji, zaczynając od takiego podziału, w którym każdy obiekt stanowi samodzielne skupienie, a kończąc na podziale, w którym wszystkie obiekty należą do jednego skupienia
- Metody k-średnich – podział populacji na z góry założoną liczbę klas. Następnie uzyskany podział jest poprawiany w ten sposób, że niektóre elementy są przenoszone do innych klas, tak, aby uzyskać minimalną wariancję wewnątrz uzyskanych klas

- Metody rozmytej analizy skupień - mogą przydzielać element do więcej niż jednej kategorii, stosowane do kategoryzacji (przydziału jednostek do jednej lub wielu kategorii)

My będziemy używać metody k-średnich wraz z funkcjami określającymi optymalną ilość klastrów dla danego zbioru.

3.2 Predykcja stanu w oparciu o przynależność do klastra.

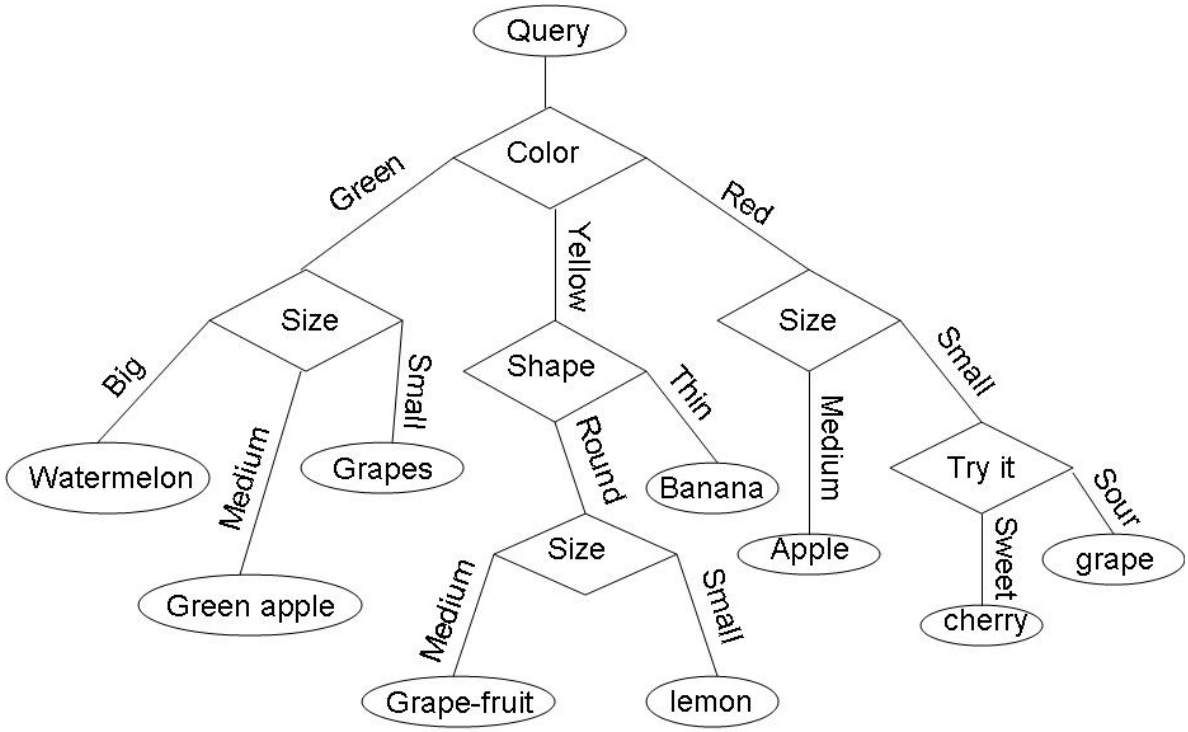
Jest to podstawowa technika predykcji przynależności do klastra. Na podstawie elementów klastra można określić jego środek. Mając dany punkt, jesteśmy w stanie porównać jego odległość od środków wyróżnionych klastrów i wybrać ten, dla którego ta odległość jest najmniejsza.

Zastosowanie w systemie:

- klastry są wyróżniane za pomocą funkcji pamk – zapisywane są punkty należące do każdego z klastrów oraz ich środki, klaster reprezentuje stan systemu
- w fazie inicjalizacji budowana jest struktura zawierająca definicję klastrów przez podanie ich środków
- w fazie wykonywania symulacji zachodzi predykcja stanu z wykorzystaniem funkcji cl_predict pakietu R

3.3 Predykcja stanu w oparciu o drzewa decyzyjne

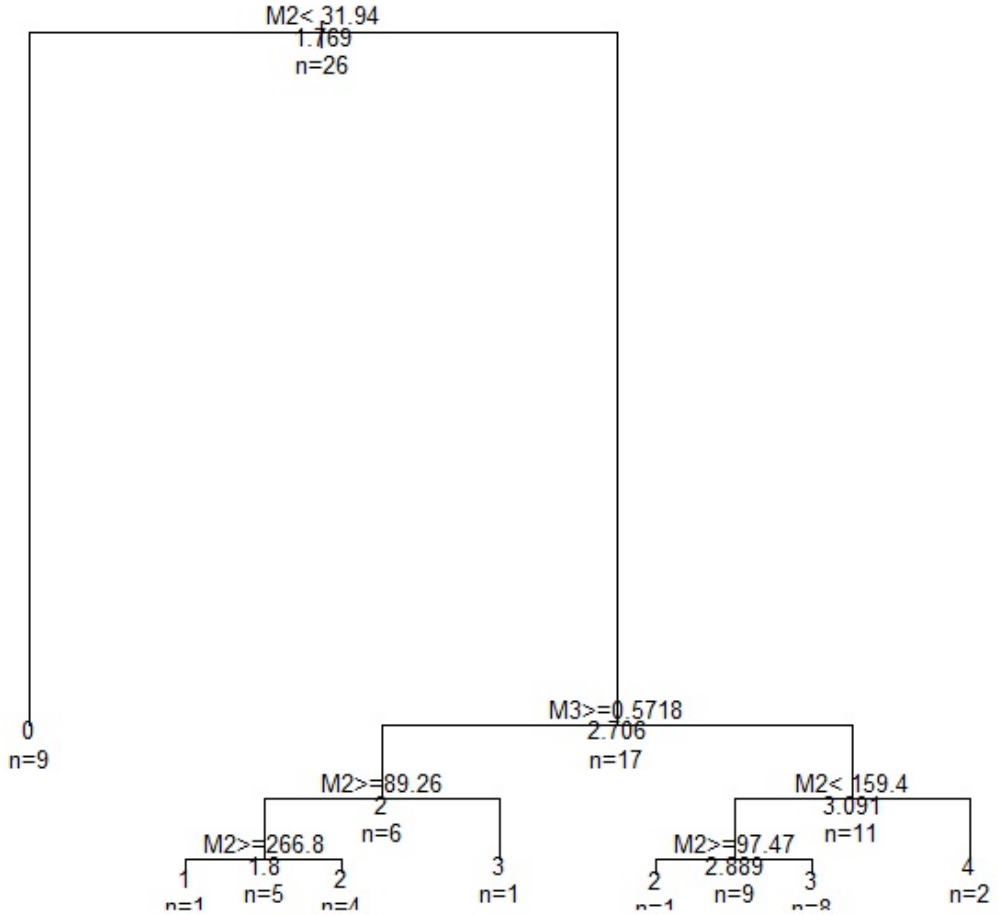
Drzewo decyzyjne jest diagramem przepływu o strukturze drzewa, gdzie każdy wierzchołek wewnętrzny oznacza testowanie atrybutu, każda krawędź reprezentuje wyjście z testu (wartość lub zbiór wartości atrybutu), a każdy liść reprezentuje klasę. W celu sklasyfikowania nieznanego obiektu wartości jego atrybutów testowane są zgodnie z informacją zawartą w drzewie decyzyjnym. W procesie testowania przechodzimy ścieżkę w drzewie od korzenia do jednego z liści – w ten sposób określana jest klasa, do której zostanie zaklasyfikowany obiekt. Rysunek poniżej prezentuje przykładowe drzewo decyzyjne. Pokazane jest na nim to, jak na podstawie drzewa decyzyjnego można sprawdzić jaki mamy owoc :



Zastosowanie w systemie:

- klastry są wyróżniane za pomocą funkcji pamk pakietu R, zapisywane są klastry oraz punkty przynależne do każdego klastra
- w fazie inicjalizacji budowane jest drzewo decyzyjne na podstawie zbioru uczącego, w naszym przypadku są to zapisane punkty z przynależnością do danego klastra
- w węzłach drzewa znajdują się warunki logiczne zbudowane na podstawie wartości pomiarów (punkty klastrów), np. M1>2 && M3<8, określają one sposób przejścia przez drzewo w sytuacji wyboru stanu dla danych testowych, jeśli warunek się zgadza wybierana jest prawa gałąź, jeśli nie – lewa
- w fazie wykonywania symulacji zachodzi predykcja stanu z wykorzystaniem drzewa zbudowanego w fazie inicjalizacji, dane testowe stanowią aktualne wartości pomiarów, stan jest określony przez wierzchołek który został wybrany po przejściu przez drzewo, wykorzystana został metoda predict pakietu R

Przykład drzewa wygenerowanego w jednej z symulacji:



Warto zauważyć, że nie wszystkie miary muszą znaleźć się w drzewie decyzyjnym. Oznacza to, że wartość tej miary nie ma wpływu na wybór stanu. Taka analiza może wspomóc proces wyboru miar użytych w symulacji.

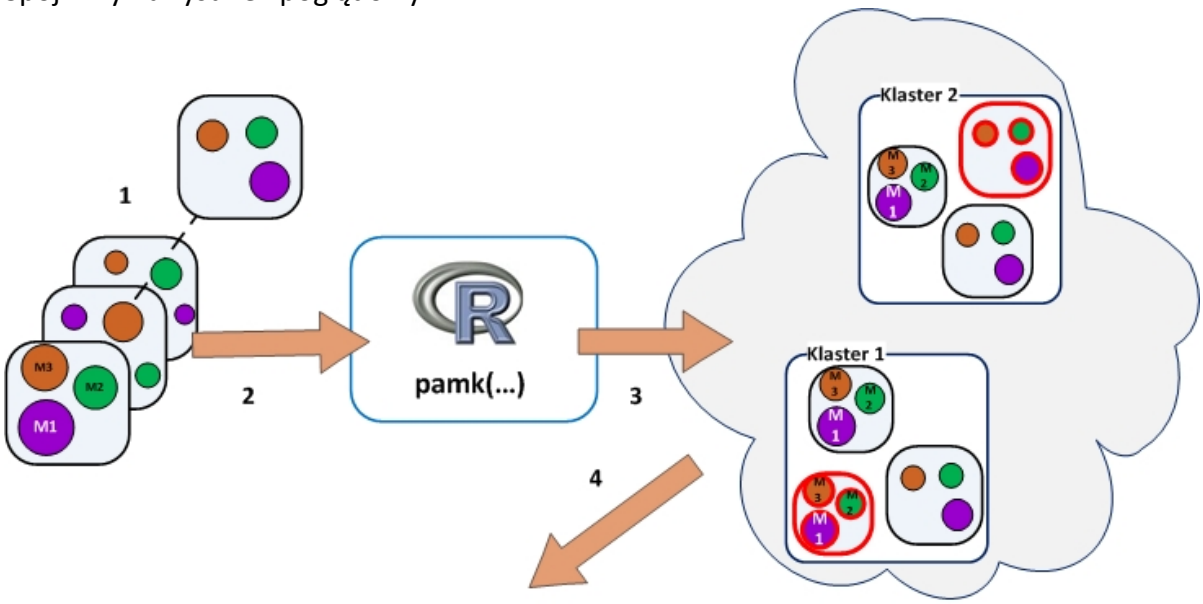
3.3 Koncepcja

Przed przejściem do implementacji ustalono cele jakie sobie stawiamy oraz wymyślono i przedyskutowano architekturę oraz sposób jej implementacji. Do implementacji wybrano algorytmy klastrujące z rodziny : k-means. W trybie nauki użyto algorytmu pamk, który jest odmianą k-means. Różni się właściwie tylko tym, że w przeciwieństwie do klasycznych k-meansów jako środki klastrów wybierane są elementy wchodzące w skład wektora wejściowego. W dodatku pamk ma bardzo użyteczną cechę – sam wyznacza optymalną ilość klastrów. Do drzew decyzyjnych użyto metod z pakietu rpart.
W tym punkcie przedstawimy ogólną koncepcję, która powstała przed przejściem do implementacji.

Tryb nauki:

- wczytywana jest konfiguracja systemu, stany, punkty należące do każdego z klastrów oraz zawartość tabeli do Qlearningu
- budowana jest struktura danych potrzebna w czasie predykcji stanu – zależnie od konfiguracji drzewo decyzyjne lub środki klastrów
- zachodzi normalna praca systemu, jeśli włączona jest eksploracja to komórki tabeli mogą się zmieniać
- w czasie pracy zapisywana jest historia pomiarów używanych w symulacji (w zależności od konfiguracji)
- po zakończeniu symulacji, przeprowadzany jest klastering, zapisywane są nowe stany systemu, przynależność punktów zapisanych w historii pomiarów do klastrów oraz jest zerowana tablica akcji
- do klastrowania używana jest metoda pamk, dolna i górna granica ilości stworzonych klastrów jest zdefiniowana przez wyrażenie podane w konfiguracji (dowolne funkcje z pakietu java.math)
- zapisywana jest nowa konfiguracja systemu, poprzednia może zostać nadpisana lub pozostać bez zmian

Spójrzmy na rysunek poglądowy:



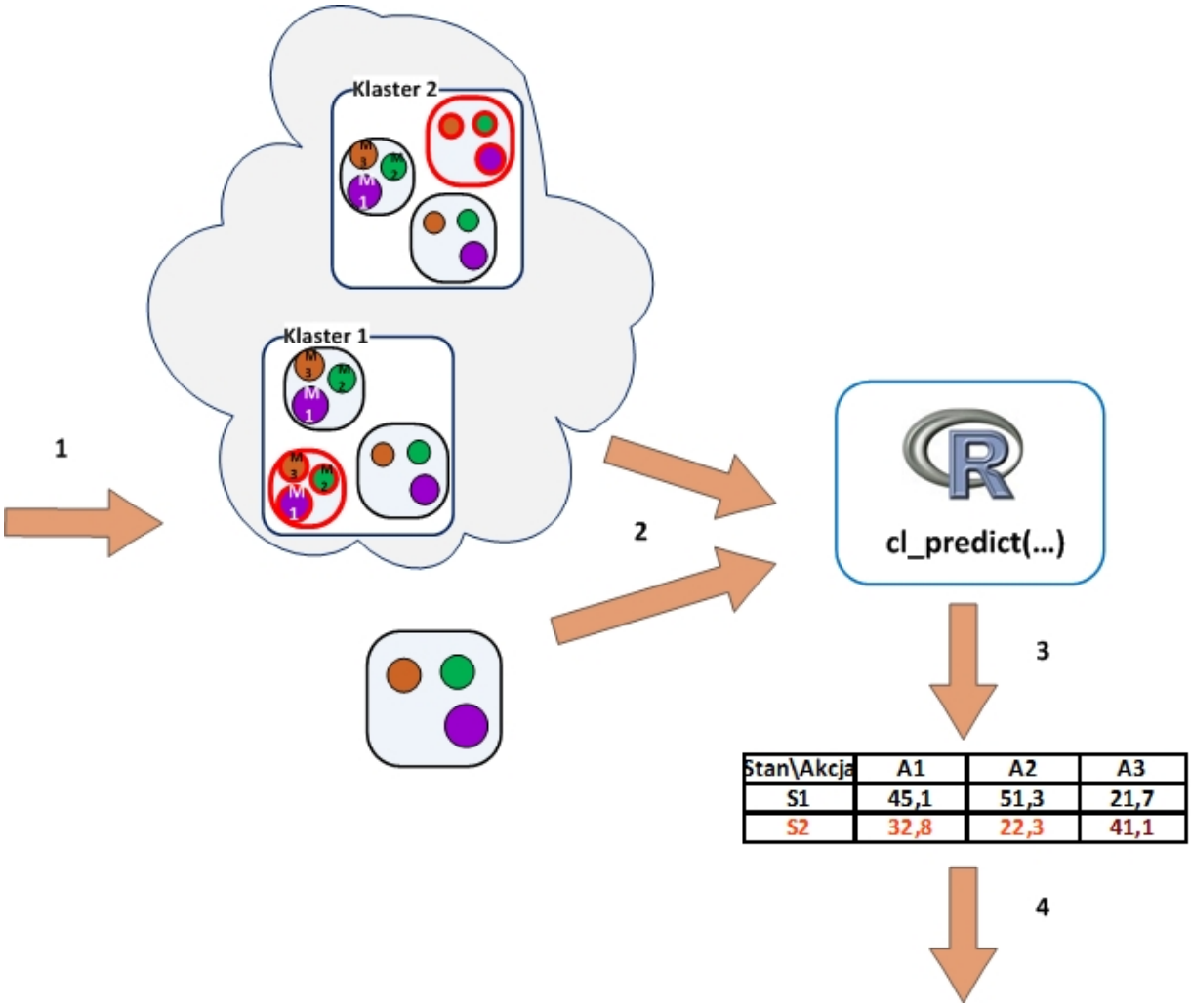
Stan\Akcja	A1	A2	A3
S1	0	0	0
S2	0	0	0

1)W kolejnych krokach symulacji, zapisywane są wartości measurementów. Jeden wpis w historii, zawiera wartości wszystkich monitorowanych wartości.
2)Po skończonej symulacji, zbiór pomiarów jest wykorzystywany do stworzenia klastrów. Są one generowane przez platformę R. Został użyty algorytm pamk – odmiana kmeans.
3)Wygenerowane klastry to zbiory pomiarów. Pojedynczy klaster jest reprezentowany przez środek. W przypadku algorytmu pamk, środkiem wybierany jest jeden z punktów klastra.
4)Klastry są zapisywane do pliku konfiguracyjnego. Dla Każdego stanu, wartości akcji zostają wyzerowane.

Tryb pracy:

- wczytywana jest konfiguracja systemu, stany, punkty należące do każdego z klastrów oraz zawartość tabeli do Qlearningu
- budowana jest struktura danych potrzebna w czasie predykcji stanu – zależnie od konfiguracji drzewo decyzyjne lub środki klastrów
- wykonuje się symulacja, predykcja stanu zachodzi w zależności od konfiguracji systemu
- w trybie eksploracji zachodzi zmiana wartości w tabeli Qlearningu

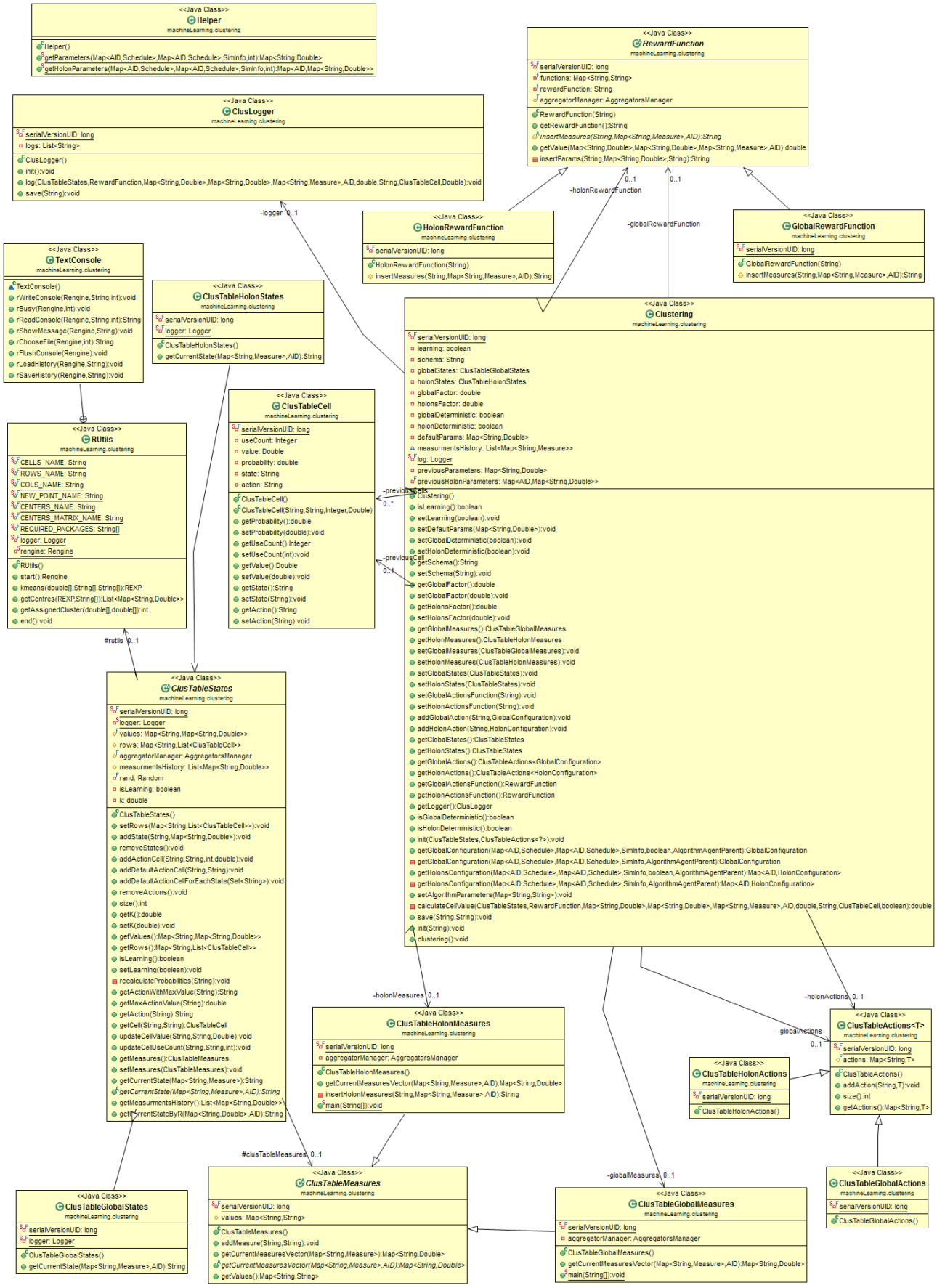
Ponownie posłużymy się rysunkiem:



1)Wykonuje się krok symulacji, w trakcie należy określić aktualny stan systemu
2)Na podstawie aktualnych wartości pomiarów wybierany jest stan systemu, wykorzystane są drzewa decyzyjne lub środki klastrów
3)Z tabeli QLearningu wybierana jest akcja do wykonania
4)Następuje kolejny krok symulacji

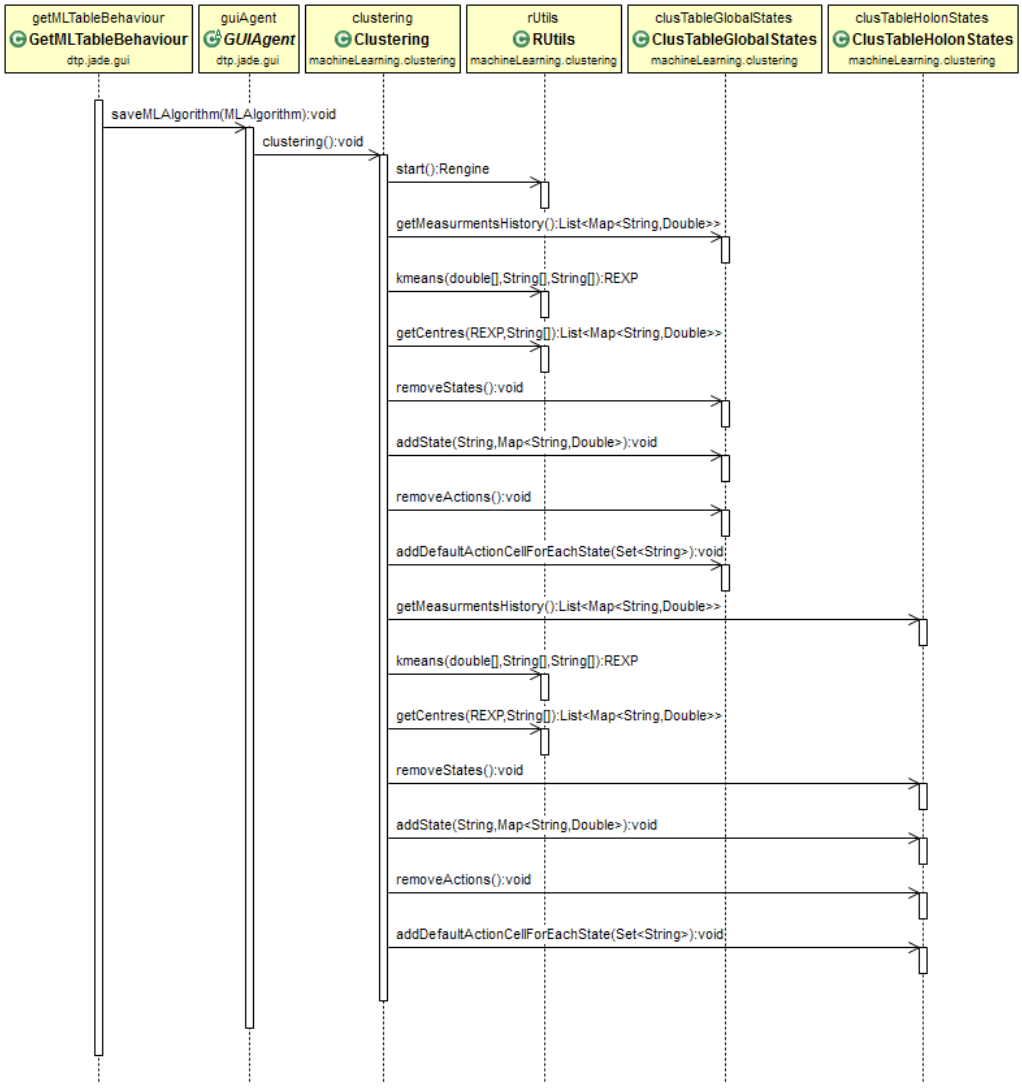
3.4 Diagram klas

Poniższy diagram klas przedstawia wszystkie klasy zaimplementowane w systemie realizujące Clustering. Podane zostały wszystkie metody oraz zależności między klasami. Przedstawione klasy można znaleźć w pakiecie machineLearning.clustering.

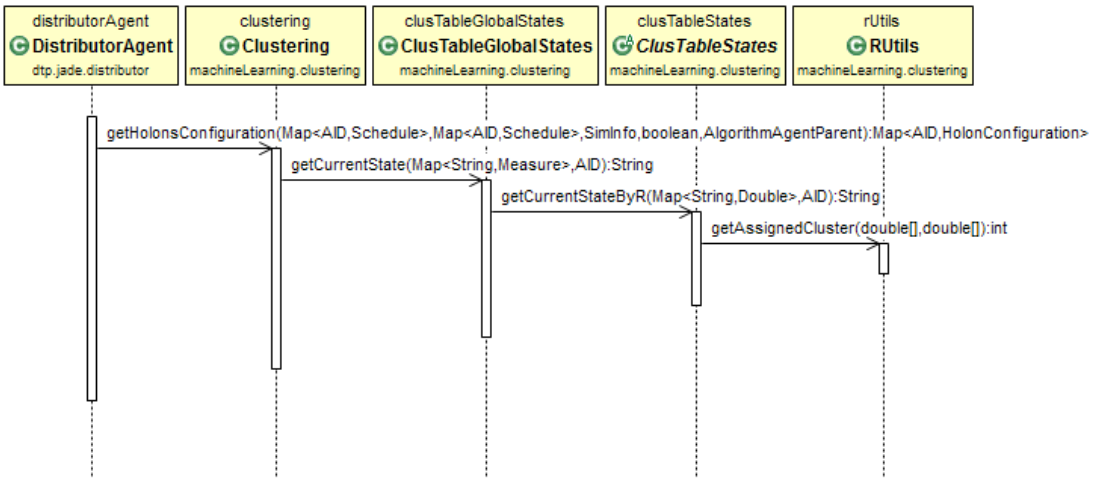


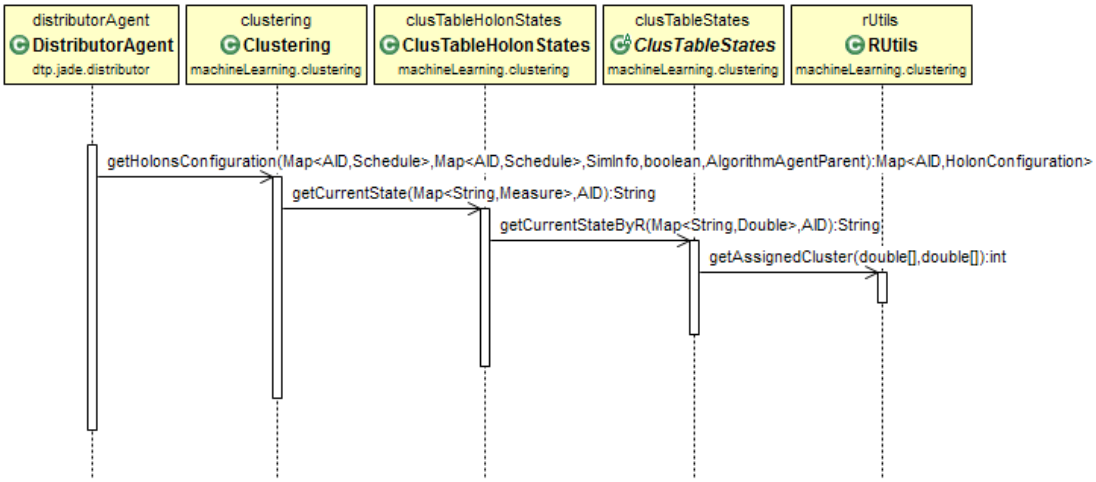
3.5 Diagramy sekwencji

Poniższy diagram sekwencji obrazuje jak dochodzi do zapisania nowego pliku konfiguracyjnego dla Clusteringu. Należy jeszcze raz wspomnieć, że ten proces wywoływany jest tylko w przypadku włączonej opcji learning (parametr learning=true). Przed zapisaniem nowej konfiguracji przeprowadzany jest clustering, cały proces jest dobrze zilustrowany na diagramie.



Poniższe dwa diagramy przedstawiają w jakiej sytuacji używany jest Clustering celem wyboru aktualnego stanu systemu. Pierwszy diagram pokazuje użycie metody cl_predict (R) w przypadku wyboru konfiguracji globalnej, podczas gdy drugi pokazuje użycie tej metody w przypadku wyboru konfiguracji lokalnej (holonicznej).





4. Podział ról w zespole

Projekt jest realizowany przez dwie osoby. Zakłada się wspólne opracowywanie i realizowanie kolejnych zagadnień, aby każdy z członków posiadał pełną wiedzę o tworzonej aplikacji. Podział zadań będzie widoczny przez przypisywanie właściciela każdego zagadnienia. Jednakże, aby spotkania dotyczące projektu przebiegały sprawnie, na kierownika projektu wybiera się Sebastiana Feduniaka. Będzie on odpowiedzialny za nadzorowanie prowadzenia projektu oraz za relacjonowanie postępu podczas spotkań z klientem.

5. Harmonogram prac

Podczas pracy nad projektem ustala się następujący harmonogram prac.

Semestr I

Termin	*Zadania*
26.03 – 9.04	Zapoznanie się z problemem PDPTW oraz systemem DispatchRider
9.04 – 23.04	Transfer wiedzy na temat systemu od Pana Piotra Pisarskiego oraz zbieranie wymagań od klienta
23.04 – 07.05	Opracowanie koncepcji systemu oraz prezentacji początkowej, podpięcie platformy R do Javy

07.05 – 14.05	Stworzenie schema dla konfiguracji, wczytywanie jej oraz zapisywanie do pliku
14.05 – 21.05	Podpięcie naszej konfiguracji pod konfigurację globalną, implementacja mockowego algorytmu clusteringu
21.05 – 28.05	Implementacja trybu uczenia
28.05 – 04.06	Implementacja trybu roboczego – wersja beta (ilość klastrów zadana)
04.06 – 11.06	Generacja optymalnej liczby klastrów
11.06 – 18.06	Testowanie – wersja końcowa
14.05 – 18.06	Dokumentacja pracy nad projektem

Semestr II

Termin	*Zadania*
01.10 – 23.10	Przegląd implementacji drzew decyzyjnych w R
23.10 – 30.10	Implementacja dodatkowych opcji konfiguracyjnych
30.10 – 13.11	Projekt systemu – możliwość wymiany algorytmu na każdym etapie pracy systemu
13.11 – 01.12	Implementacja drzew decyzyjnych
01.12 – 01.01	Testy poprawności oraz porównawcze z poprzednim rozwiązaniem
01.01 – 29.01	Testy do wyboru miar

6. Konfiguracja systemu

6.1 Instalacja oraz konfiguracja platformy R

Nasz algorytm uczenia maszynowego do prawidłowej pracy wymaga zainstalowanego pakietu R oraz dodatkowego pluginu dla Javy, który umożliwia korzystanie z R z poziomu kodu Javy.
Poniżej lista kroków wymagana do poprawnej instalacji pakietu R na system Windows :

1. Należy pobrać aktualną wersję platformy R ze strony :
http://r.meteo.uni.wroc.pl/ -> wybrać „Download R for Windows” -> wybrać “base” -> wybrać link do pobierania

2. Zainstalować pakiet R – uruchomić instalator oraz przejść wszystkie jego kroki

3. Zainstalować plugin rJava :
Po instalacji powinien się pojawić skrót do R. Uruchomić R w wersji adekwatnej do posiadanego systemu : R – wersja 32 bitowa, R x64 – wersja 64 bitowa.
Wybrać mirror – packages -> choose CRAN-Mirror -> wybrać „Wrocław”
Wybrać repozytoria – packages -> repositories -> zaznaczyć wszystkie
Zainstalować rJava – w linii poleceń R wprowdzić install.packages(„rJava”) -> wybrać mirror, najlepiej Wrocław -> instalator powinien zainstalować pakiet.

4. Zainstalować pakiet fpc – w linii poleceń R wprowadzić install.packages(„fpc”) -> wybrać mirror, najlepiej Wrocław -> instalator powinien zainstalować pakiet.

5. Podobnie jak powyżej zainstalować pakiet clue.

6. Konfiguracja ścieżek systemowych :
R_HOME – ścieżka do katalogu gdzie zainstalowano R, np. D:\studia\R\R-2.12.2
LD_LIBRARY_PATH – dodać %R_HOME%\library\rJava\jri
PATH – dodać %R_HOME%\library\rJava\jri, %R_HOME%\bin, %R_HOME%\bin\x64 <- w przypadku systemów 64-bitowych, %R_HOME%\bin\i386 <- w przypadku systemów 32-bitowych

7. Skonfigurować system system tak jak w punkcie 6.2 (tryb nauki – learning=true) – uruchomić symulację i sprawdzić w logach czy R uruchamia się bez błędów. Na końcu symulacji

6.2 Użycie naszego algorytmu w systemie

Aby włączyć nasz algorytm tak aby był używany w „Dispatch Rider” należy zmienić główną konfigurację : „configuration.xml”. Jako algorithm należy ustawić „Clustering”, jako file – ścieżkę do pliku opisującego tabelę dla klastryzacji, exploration – true gdy zmieniamy wartości tabeli, false gdy nie zmieniamy.
Przykład :

```
...
        <mlAlgorithm file="clustable.xml" algorithm="Clustering" exploration="false" >
        <param name="bestDist" value="828.94"/>
    </mlAlgorithm>
...
```

Możemy również określać dodatkowe parametry (stałe), które możemy potem używać np. w funkcji nagrody.

7. Opis implementacji

7.1 Plik konfiguracyjny

Na potrzeby projektu stworzono własną schemę XSD, która opisuje jak powinna wyglądać konfiguracja naszego systemu. Są w niej opisane następujące rzeczy :

- atrybuty tagu ClusTable – learning: true – tryb uczenia, false – tryb pracy; minClustCount – funkcja do obliczenia dolnej granicy ilości klastrów, maxClustCount – funkcja do obliczenia górnej granicy ilości klastrów, overwriteConf – true określa że nowa konfiguracja nadpisze poprzednią, useTree – określa że zostaną użyte drzewa decyzyjne, usepam – określa algorytm użyty w pamk
- globalne measurementy – używane do wyznaczenia stanów globalnych
- lokalne measurementy – używane do wyznaczenia stanów holonów
- globalne stany
- holoniczne stany
- globalne akcje
- holoniczne stany

- globalny content – wartości komórek tabeli globalnej
- lokalny content – wartości komórek tabeli dla holonów
- globalne obserwacje – przypisanie wartości globalnych pomiarów do globalnych stanów
- lokalne obserwacje – przypisanie wartości pomiarów holonów do stanów holonów

Schema :

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

	<xs:element name="ClusTable">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="structure" type="structureType"
					minOccurs="1" maxOccurs="1" />
				<xs:element name="content" minOccurs="0" maxOccurs="1">
					<xs:complexType>
						<xs:sequence>
							<xs:element name="globalTableContent"
type="contentType"
								minOccurs="0" maxOccurs="1" />
							<xs:element name="holonTableContent"
type="contentType"
								minOccurs="0" maxOccurs="1" />
						</xs:sequence>
					</xs:complexType>
				</xs:element>
				<xs:element name="observations" minOccurs="1" maxOccurs="1"
					type="globalHolonObservationsType" />
			</xs:sequence>
			<xs:attribute name="learning" type="xs:boolean" default="false" />
			<xs:attribute name="useTrees" type="xs:boolean" default="false" />
			<xs:attribute name="minClustCount" type="xs:string"
				default="2" />
			<xs:attribute name="maxClustCount" type="xs:string"
				default="sqrt(N)" />
			<xs:attribute name="usepam" type="xs:boolean" default="true" />
			<xs:attribute name="overwriteConf" type="xs:boolean"
				default="true" />
		</xs:complexType>
	</xs:element>

	<xs:complexType name="structureType">
		<xs:sequence>
			<xs:element name="globalMeasures" type="measuresType"
				minOccurs="0" maxOccurs="1" />
			<xs:element name="globalStates" type="statesType"
				minOccurs="0" maxOccurs="1" />
			<xs:element name="globalActions" type="actionsType"
				minOccurs="0" maxOccurs="1" />
			<xs:element name="holonMeasures" type="measuresType"
				minOccurs="0" maxOccurs="1" />
			<xs:element name="holonStates" type="statesType"
				minOccurs="0" maxOccurs="1" />
			<xs:element name="holonActions" type="holonActionsType"
				minOccurs="0" maxOccurs="1" />
		</xs:sequence>
	</xs:complexType>

	<xs:complexType name="measuresType">
```

```
            <xs:sequence>
                  <xs:element name="measure" minOccurs="1" maxOccurs="unbounded"
                        type="measureType" />
            </xs:sequence>
      </xs:complexType>

      <xs:complexType name="measureType">
            <xs:attribute name="name" type="xs:string" use="required" />
            <xs:attribute name="value" type="xs:string" use="required" />
      </xs:complexType>

      <xs:complexType name="statesType">
            <xs:sequence>
                  <xs:element name="state" minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="measure" minOccurs="0"
maxOccurs="unbounded">
                                          <xs:complexType>
                                                <xs:attribute name="name"
type="xs:string" use="required" />
                                                <xs:attribute name="value"
type="xs:string" use="required" />
                                          </xs:complexType>
                                    </xs:element>
                              </xs:sequence>
                              <xs:attribute name="name" />
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
            <xs:attribute name="k" type="xs:double" use="required" />
      </xs:complexType>

      <xs:complexType name="actionsType">
            <xs:sequence>
                  <xs:element name="action" minOccurs="1" maxOccurs="unbounded">
                        <xs:complexType>
                              <xs:attribute name="name" type="xs:string"
use="required" />
                              <xs:attribute name="sendingType" type="xs:string"
                                    default="null" />
                              <xs:attribute name="choosingByCost" type="xs:string"
                                    default="null" />
                              <xs:attribute name="simmulatedTrading" type="xs:int"
                                    default="-1" />
                              <xs:attribute name="STDepth" type="xs:int" default="-1"
/>
                              <xs:attribute name="chooseWorstCommission"
type="xs:string"
                                    default="null" />
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
            <xs:attribute name="function" type="xs:string" use="required" />
            <xs:attribute name="factor" type="xs:double" use="required" />
            <xs:attribute name="deterministic" type="xs:boolean"
                  default="true" />
      </xs:complexType>


      <xs:complexType name="holonActionsType">
            <xs:sequence>
```

```
                  <xs:element name="action" minOccurs="1" maxOccurs="unbounded">
                        <xs:complexType>
                              <xs:attribute name="name" type="xs:string"
use="required" />
                              <xs:attribute name="algorithm" type="xs:string"
                                    default="null" />
                              <xs:attribute name="newCommissionCostByDist"
type="xs:string"
                                    default="null" />
                              <xs:attribute name="simmulatedTrading" type="xs:string"
                                    default="null" />
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
            <xs:attribute name="function" type="xs:string" use="required" />
            <xs:attribute name="factor" type="xs:double" use="required" />
            <xs:attribute name="deterministic" type="xs:boolean"
                  default="true" />
      </xs:complexType>

      <xs:complexType name="contentType">
            <xs:sequence>
                  <xs:element name="state" minOccurs="1" maxOccurs="unbounded">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="action" minOccurs="1"
maxOccurs="unbounded">
                                          <xs:complexType>
                                                <xs:attribute name="name"
type="xs:string" use="required" />
                                                <xs:attribute name="useCount"
type="xs:int" use="required" />
                                                <xs:attribute name="value"
type="xs:double" use="required" />
                                          </xs:complexType>
                                    </xs:element>
                              </xs:sequence>
                              <xs:attribute name="name" type="xs:string"
use="required" />
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
      </xs:complexType>


      <xs:complexType name="globalHolonObservationsType">
            <xs:sequence>
                  <xs:element name="globalObservations" type="observationsType"
                        minOccurs="1" maxOccurs="1" />
                  <xs:element name="holonObservations" type="observationsType"
                        minOccurs="1" maxOccurs="1" />
            </xs:sequence>
      </xs:complexType>

      <xs:complexType name="observationsType">
            <xs:sequence>
                  <xs:element name="observation" minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="measure" type="measureType"
                                          minOccurs="1" maxOccurs="unbounded" />
                              </xs:sequence>
```

```
                                <xs:attribute name="state" type="xs:string"
use="required" />
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>

</xs:schema>
```

Przykładowy plik konfiguracyjny :

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ClusTable xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" learning="true"
maxClustCount="sqrt(N)" minClustCount="2" overwriteConf="true" useTrees="true"
usepam="true" xsi:noNamespaceSchemaLocation="xml/schemes/clustable.xsd">
    <structure>
        <globalMeasures>
            <measure name="M1" value="avg(AverageMinDistBetweenAllCommissions)"/>
            <measure name="M2" value="avg(WaitTime)"/>
            <measure name="M3"
value="avg(AverageNumberOfComsWithinTimeWinOfAllCommissions)"/>
        </globalMeasures>
        <globalStates k="0.5">
            <state name="S1">
                <measure name="M3" value="18.05"/>
                <measure name="M2" value="4.98809692479999"/>
                <measure name="M1" value="3.4717238964833186"/>
            </state>
            <state name="S0">
                <measure name="M3" value="83.04069767441861"/>
                <measure name="M2" value="0.0"/>
                <measure name="M1" value="4.541293127503809"/>
            </state>
        </globalStates>
        <globalActions factor="0.9"
function="2*_holonsCount/holonsCount+bestDist/dist+_costOfCommission/costOfCommission">
            <action STDepth="0" choosingByCost="false" name="A2"
simmulatedTrading="0"/>
            <action STDepth="0" choosingByCost="true" name="A1" simmulatedTrading="0"/>
            <action STDepth="0" choosingByCost="false" name="A4"
simmulatedTrading="1"/>
            <action STDepth="0" choosingByCost="true" name="A3" simmulatedTrading="1"/>
            <action STDepth="3" choosingByCost="false" name="A6"
simmulatedTrading="0"/>
            <action STDepth="3" choosingByCost="true" name="A5" simmulatedTrading="0"/>
            <action STDepth="3" choosingByCost="false" name="A8"
simmulatedTrading="1"/>
            <action STDepth="3" choosingByCost="true" name="A7" simmulatedTrading="1"/>
            <action STDepth="8" choosingByCost="true" name="A11"
simmulatedTrading="1"/>
            <action STDepth="8" choosingByCost="false" name="A10"
simmulatedTrading="0"/>
            <action STDepth="8" choosingByCost="true" name="A9" simmulatedTrading="0"/>
            <action STDepth="8" choosingByCost="false" name="A12"
simmulatedTrading="1"/>
        </globalActions>
        <holonMeasures>
            <measure name="M1" value="AverageMinDistBetweenUndeliveredCommissions"/>
            <measure name="M2" value="WaitTime"/>
            <measure name="M3"
value="AverageNumberOfComsWithinTimeWinOfAllCommissions"/>
```

```
        </holonMeasures>
        <holonStates k="0.5">
            <state name="S1">
                <measure name="M3" value="18.05"/>
                <measure name="M2" value="4.98809692479999"/>
                <measure name="M1" value="3.4717238964833186"/>
            </state>
            <state name="S0">
                <measure name="M3" value="83.04069767441861"/>
                <measure name="M2" value="0.0"/>
                <measure name="M1" value="4.541293127503809"/>
            </state>
        </holonStates>
        <holonActions factor="0.9"
function="2*_holonCommissions/holonCommissions+_costOfCommission/costOfCommission">
            <action algorithm="BruteForceAlgorithm" name="A2"
simmulatedTrading="true"/>
            <action algorithm="BruteForceAlgorithm" name="A1"
simmulatedTrading="false"/>
            <action algorithm="BruteForceAlgorithm2" name="A4"
simmulatedTrading="true"/>
            <action algorithm="BruteForceAlgorithm2" name="A3"
simmulatedTrading="false"/>
        </holonActions>
    </structure>
    <content>
        <globalTableContent>
            <state name="S1">
                <action name="A2" useCount="0" value="0.0"/>
                <action name="A1" useCount="0" value="0.0"/>
                <action name="A4" useCount="0" value="0.0"/>
                <action name="A3" useCount="0" value="0.0"/>
                <action name="A6" useCount="0" value="0.0"/>
                <action name="A5" useCount="0" value="0.0"/>
                <action name="A8" useCount="0" value="0.0"/>
                <action name="A7" useCount="0" value="0.0"/>
                <action name="A11" useCount="0" value="0.0"/>
                <action name="A10" useCount="0" value="0.0"/>
                <action name="A9" useCount="0" value="0.0"/>
                <action name="A12" useCount="0" value="0.0"/>
            </state>
            <state name="S0">
                <action name="A2" useCount="0" value="0.0"/>
                <action name="A1" useCount="0" value="0.0"/>
                <action name="A4" useCount="0" value="0.0"/>
                <action name="A3" useCount="0" value="0.0"/>
                <action name="A6" useCount="0" value="0.0"/>
                <action name="A5" useCount="0" value="0.0"/>
                <action name="A8" useCount="0" value="0.0"/>
                <action name="A7" useCount="0" value="0.0"/>
                <action name="A11" useCount="0" value="0.0"/>
                <action name="A10" useCount="0" value="0.0"/>
                <action name="A9" useCount="0" value="0.0"/>
                <action name="A12" useCount="0" value="0.0"/>
            </state>
        </globalTableContent>
        <holonTableContent>
            <state name="S1">
                <action name="A2" useCount="0" value="0.0"/>
                <action name="A1" useCount="0" value="0.0"/>
                <action name="A4" useCount="0" value="0.0"/>
                <action name="A3" useCount="0" value="0.0"/>
```

```
        </state>
        <state name="S0">
            <action name="A2" useCount="0" value="0.0"/>
            <action name="A1" useCount="0" value="0.0"/>
            <action name="A4" useCount="0" value="0.0"/>
            <action name="A3" useCount="0" value="0.0"/>
        </state>
    </holonTableContent>
</content>
<observations>
    <globalObservations>
        <observation state="S0">
            <measure name="M1" value="4.739013395132939"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="89.88172043010752"/>
        </observation>
        <observation state="S0">
            <measure name="M1" value="4.705084539844029"/>
            <measure name="M2" value="53.15685459146846"/>
            <measure name="M3" value="88.01648351648352"/>
        </observation>
        <observation state="S0">
            <measure name="M1" value="4.5329702962403635"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="84.06896551724138"/>
        </observation>
        <observation state="S0">
            <measure name="M1" value="4.541293127503809"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="83.04069767441861"/>
        </observation>
        <observation state="S0">
            <measure name="M1" value="4.541293127503809"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="83.04069767441861"/>
        </observation>
        <observation state="S0">
            <measure name="M1" value="4.541293127503809"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="83.04069767441861"/>
        </observation>
        <observation state="S0">
            <measure name="M1" value="4.520275872888202"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="74.12337662337663"/>
        </observation>
        <observation state="S0">
            <measure name="M1" value="4.579753187005152"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="73.09868421052632"/>
        </observation>
        <observation state="S0">
            <measure name="M1" value="4.450152746185214"/>
            <measure name="M2" value="33.03237049098179"/>
            <measure name="M3" value="62.276923076923076"/>
        </observation>
        <observation state="S0">
            <measure name="M1" value="4.522224943323062"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="52.25454545454546"/>
        </observation>
        <observation state="S1">
```

```
            <measure name="M1" value="4.455859469753977"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="42.79347826086956"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="4.261656181315167"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="35.87179487179487"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="3.8450719070358925"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="32.93055555555556"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="3.972966984515593"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="31.058823529411764"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="3.2533720244921693"/>
            <measure name="M2" value="16.675444679663244"/>
            <measure name="M3" value="19.90909090909091"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="3.4717238964833186"/>
            <measure name="M2" value="4.98809692479999"/>
            <measure name="M3" value="18.05"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="2.614723856711792"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="7.285714285714286"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="3.050511166163757"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="6.333333333333333"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="3.660613399396509"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="5.0"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="1.8027756377319946"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="1.75"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="4.123105625617661"/>
            <measure name="M2" value="65.23044737829953"/>
            <measure name="M3" value="0.5"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="2.23606797749979"/>
            <measure name="M2" value="32.19903382206219"/>
            <measure name="M3" value="0.0"/>
        </observation>
    </globalObservations>
    <holonObservations>
        <observation state="S0">
```

```
        <measure name="M1" value="4.739013395132939"/>
        <measure name="M2" value="0.0"/>
        <measure name="M3" value="89.88172043010752"/>
    </observation>
    <observation state="S0">
        <measure name="M1" value="4.705084539844029"/>
        <measure name="M2" value="53.15685459146846"/>
        <measure name="M3" value="88.01648351648352"/>
    </observation>
    <observation state="S0">
        <measure name="M1" value="4.5329702962403635"/>
        <measure name="M2" value="0.0"/>
        <measure name="M3" value="84.06896551724138"/>
    </observation>
    <observation state="S0">
        <measure name="M1" value="4.541293127503809"/>
        <measure name="M2" value="0.0"/>
        <measure name="M3" value="83.04069767441861"/>
    </observation>
    <observation state="S0">
        <measure name="M1" value="4.541293127503809"/>
        <measure name="M2" value="0.0"/>
        <measure name="M3" value="83.04069767441861"/>
    </observation>
    <observation state="S0">
        <measure name="M1" value="4.541293127503809"/>
        <measure name="M2" value="0.0"/>
        <measure name="M3" value="83.04069767441861"/>
    </observation>
    <observation state="S0">
        <measure name="M1" value="4.520275872888202"/>
        <measure name="M2" value="0.0"/>
        <measure name="M3" value="74.12337662337663"/>
    </observation>
    <observation state="S0">
        <measure name="M1" value="4.579753187005152"/>
        <measure name="M2" value="0.0"/>
        <measure name="M3" value="73.09868421052632"/>
    </observation>
    <observation state="S0">
        <measure name="M1" value="4.450152746185214"/>
        <measure name="M2" value="33.03237049098179"/>
        <measure name="M3" value="62.276923076923076"/>
    </observation>
    <observation state="S0">
        <measure name="M1" value="4.522224943323062"/>
        <measure name="M2" value="0.0"/>
        <measure name="M3" value="52.25454545454546"/>
    </observation>
    <observation state="S1">
        <measure name="M1" value="4.455859469753977"/>
        <measure name="M2" value="0.0"/>
        <measure name="M3" value="42.79347826086956"/>
    </observation>
    <observation state="S1">
        <measure name="M1" value="4.261656181315167"/>
        <measure name="M2" value="0.0"/>
        <measure name="M3" value="35.87179487179487"/>
    </observation>
    <observation state="S1">
        <measure name="M1" value="3.8450719070358925"/>
        <measure name="M2" value="0.0"/>
```

```
            <measure name="M3" value="32.93055555555556"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="3.972966984515593"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="31.058823529411764"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="3.2533720244921693"/>
            <measure name="M2" value="16.675444679663244"/>
            <measure name="M3" value="19.90909090909091"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="3.4717238964833186"/>
            <measure name="M2" value="4.98809692479999"/>
            <measure name="M3" value="18.05"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="2.6147238567117 92"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="7.285714285714286"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="3.050511166163757"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="6.333333333333333"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="3.660613399396509"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="5.0"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="1.8027756377319946"/>
            <measure name="M2" value="0.0"/>
            <measure name="M3" value="1.75"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="0.0"/>
            <measure name="M2" value="65.23044737829953"/>
            <measure name="M3" value="0.5"/>
        </observation>
        <observation state="S1">
            <measure name="M1" value="0.0"/>
            <measure name="M2" value="32.19903382206219"/>
            <measure name="M3" value="0.0"/>
        </observation>
    </holonObservations>
</observations>
</ClusTable>
```

7.2 Zapisywanie oraz wczytywanie konfiguracji

W pakiecie machineLearning.xml stworzyliśmy dwie klasy, jedną do wczytywania konfiguracji (ClustableStructureParser.java), a drugą do jej zapisywania (ClusTableToXmlWriter.java).

ClustableStructureParser posiada metodę statyczną parę :

```
public static void parse(String filename, Clustering table)
                        throws ParseException
```

Która jest wykonywana na obiekcie this przy inicjalizacji algorytmu clusteringu :

Klasa Clustering :

```
@Override
        public void init(String fileName) {
                try {
                        ClusTableStructureParser.parse(fileName, this);
                } catch (ParseException e) {
                        e.printStackTrace();
                        System.exit(0);
                }

        }
```

ClustableToXmlWriter ma statyczną metodę writeToXml, która zapisuje ClusTable do pliku xml :

```
public static void writeToXML(String fileName, Clustering table)
                        throws Exception
```

Konfiguracja jest zapisywana do pliku w trzech przypadkach :

- learning = true, exploration = true
- learning = true, exploration = false
- learning = false, exploration = true

Wówczas nadpisywany jest plik (w zależności od konfiguracji), który został na początku wczytywany, jak również zapisywana jest kopia w folderze, z którego zostały pobrane dane testowe, tag configuration w konfiguracji głównej.

Obie klasy do operacjach na XML'ach wykorzystują bibliotekę DOM.

7.3 Moduł uczenia maszynowego – clustering

Poniżej zostaną opisane zaimplementowane klasy, które wchodzą w skład pakietu machineLearning.clustering. Jest to szereg klas implementujących nasz algorytm. Główną klasą w tym pakiecie jest Clustering rozszerzająca MLAlgorithm.

1. ClusLogger.java

Klasa zbierająca informację na temat symulacji. Przy końcu symulacji dane te zapisywane są do pliku z logami. Ich zawartość może być przydatna przy analizie działania systemu.

Metoda do logowania :

```
public void log(ClusTableStates states, RewardFunction function,
                Map<String, Double> parameters, Map<String, Double> prevParameters,
                Map<String, Measure> measures, AID aid, double factor,
                String currentState, ClusTableCell previousCell, Double value)
```

Metoda wywoływana w metodzie save klasy Clustering przy końcu symulacji.

```
public void save(String fileName) throws Exception
```

2. ClusTableActions.java

Klasa przechowywująca akcje mogące występować podczas działania systemu – zmiany konfiguracji.

Mapuje nazwe akcji na akcję. T – konfiguracja do użycia przy zmianie tej akcji, lokalna lub globalna.

```
protected final Map<String, T> actions = new HashMap<String, T>();
```

Dodaje akcje

```
public void addAction(String name, T conf)
```

Zwraca ilość akcji

```
public int size()
```

Zwraca akcje

```
public Map<String, T> getActions()
```

3. ClusTableGlobalActions.java

Rozszerza ClusTableActions. Używa jako akcji konfiguracji globalnej.

```
public class ClusTableGlobalActions extends ClusTableActions<GlobalConfiguration>
```

4. ClusTableHolonActions.java

Rozszerza ClusTableActions. Używa jako akcji konfiguracji holonów.

```
public class ClusTableHolonActions extends ClusTableActions<HolonConfiguration>
```

5. ClusTableCell.java

Reprezentuje komórkę w tabeli clustable. Oznacza ile dla danego stanu i danej akcji wynosi wartość komórki oraz ile razy ta komórka została użyta.

Pola oraz konstruktor. Czyli ile wynosi wartość oraz ilość użyć dla zadanej akcji oraz stanu.

```
private Integer useCount = 0;
private Double value = 0.0;
private double probability;
private String state;
private String action;

public ClusTableCell() {

}

public ClusTableCell(String state, String action, Integer useCount,
                     Double value) {
        super();
        this.useCount = useCount;
        this.value = value;
        this.state = state;
        this.action = action;
}
```

Zwraca prawdopodobieństwo wykonania akcji.

```
public double getProbability()
```

Ustawia wyliczone prawdopodobieństwo wykonania akcji w stanie.

```
public void setProbability(double probability)
```

Zwraca ile razy do tej pory ta komórka została użyta.

```
public Integer getUseCount()
```

Ustawia ilość użyć tej komórki.

```
public void setUseCount(int useCount)
```

Zwraca wartość komórki.

```
public Double getValue()
```

Ustawia wartość komórki

```
public void setValue(double value)
```

Zwraca, dla którego stanu jest ta komórka

```
public String getState()
```

Ustawia stan dla tej komórki

```
public void setState(String state)
```

Zwraca akcję dla tej komórki

```
public String getAction()
```

Ustawia akcję dla tej komórki

```
public void setAction(String action)
```

6. ClusTableMeasures.java

Reprezentuje measurmenty używane przez nasz system do reprezentowania stanów. Każdy stan jest widziany jako wektor wartości measurmentów. Przy czym mamy measurmeny używane dla stanów globalnych oraz takie używane dla stanów holonicznych.

Mapuje nazwę measurmentu na funkcję w jaki sposób się wylicza ten measurment. Np. M1 -> avg(AvarageMinDistanceBetweenAllComissions)

```
protected Map<String, String> values = new TreeMap<String, String>();
```

Dodaje measurment

```
public void addMeasure(String key, String value)
```

Metoda abstrakcyjna implementowana przez measurmenty globalne oraz lokalne. Zwraca wartości wyliczone wartości measurmentów. Czyli mając nazwę measurmentu oraz funkcję do jej obliczania funkcja ta powinna zwrócić wartość takiego measurmentu dla takich zebranych measurmentów. Np. M1 -> 15.045.

```
public abstract Map<String, Double> getCurrentMeasuresVector(
                    Map<String, Measure> measures, AID aid);
```

7. ClusTableGlobalMeasures.java

Reprezentuje measurmenty globalne używane w naszym systemie. Są to przeważnie wartości zbierane per holon, które są agregowane którąś z funkcji agregujących.

Dla measurmentów, które mają w sobie funkcje agregujące – potrafi na podstawie stringowej reprezentacji zwrócić zagregowaną wartość.

```
private AggregatorsManager aggregatorManager = new AggregatorsManager();
```

Dla podanych parametrów symulacji (measures) zwraca wektor wartości dla naszych measurmentów : nazwa measurmentu -> wartość measurmentu

```
@Override
public Map<String, Double> getCurrentMeasuresVector(
                Map<String, Measure> measures, AID aid)
```

8. ClusTableHolonMeasures.java

Reprezentuje measurmenty dla holonów. Przedstawia jakie measurmenty będą brane pod uwagę przy wyliczaniu stanów holonicznych.

Dla podanych parametrów symulacji (measures) zwraca wektor wartości dla naszych measurmentów : nazwa measurmentu -> wartość measurmentu. Wektor measurmentów jest zwracany dla konkretnego agenta, o którego się w danym momencie pytamy (aid).

```
@Override
public Map<String, Double> getCurrentMeasuresVector(
                Map<String, Measure> measures, AID aid)
```

Oblicza podaną funkcję (fun) dla danego agenta (aid)

```
private String insertHolonMeasures(String fun,
                Map<String, Measure> measures, AID aid)
```

9. ClusTableStates.java

Reprezentuje stany systemu. Czyli to co chcemy generować I to do czego chcemy dopasowywać measurmenty. Klasa abstrakcyjna mająca dwie implementację – globalną oraz holoniczną.

Mapuje nazwę stanu na nazwy measurmentów, które z kolei są mapowane na ich wartości, np. :

S1 -> M1 -> 0.1

M2 -> 2.3

M3 -> 0.98

S2 -> M1 -> 0.13

M2 -> 5.3

M3 -> 3.38

S3 -> …

```
protected final Map<String, Map<String, Double>> values = new HashMap<String, Map<String, Double>>();
```

Mapuje nazwę stanu na komórki w tabeli, które jej dotyczą. Jest to właściwie tabela.

```
protected Map<String, List<ClusTableCell>> rows = new HashMap<String, List<ClusTable-Cell>>();
```

Dla measurmentów, które mają w sobie funkcje agregujące – potrafi na podstawie stringowej re-prezentacji zwrócić zagregowaną wartość.

```
protected AggregatorsManager aggregatorManager = new AggregatorsManager();
```

Measurmenty używane do generacji tych stanów.

```
protected ClusTableMeasures clusTableMeasures;
```

Historia measurmentów. Trzyma listę dotychczaszowo obliczonych measurmentów, np. :

[(M1=2, M2=3, M3=5), (M1=8, M2=12, M3=6)...]

```
protected List<Map<String, Double>> measurmentsHistory = new LinkedList<Map<String, Double>>();
```

Mówi czy jest włączony tryb uczenia

```
private boolean isLearning;
```

Stała używana do wyliczania prawdopodobieństwa użycia akcji. Podawana w konfiguracji tabeli.

```
private double k;
```

Klasa wrapująca silnik R – używany do clustering

```
protected static RUtils rutils;
```

Tworzy oraz inicjalizuje R

```
public ClusTableStates() {
	rutils = new RUtils();
	rutils.start();
}
```

Ustawia wiersze tabeli.

```
public void setRows(Map<String, List<ClusTableCell>> rows) {
	this.rows = rows;
}
```

Dodaje stan, czyli jego nazwę I identyfikujący go wektor measurmentów

```
public void addState(String name, Map<String, Double> value) {
	values.put(name, value);
}
```

Usuwa stany

```
public void removeStates() {
	values.clear();
}
```

Dodaje komórkę, w tabeli. Dla określonego stanu i akcji.

```
public void addActionCell(String stateName, String actionName,
		int useCount, double value)
```

Dodaje domyślną komórkę do tabeli. Używany w trybie nauki aby uzupełnić tabelę zerami.

```
public void addDefaultActionCell(String stateName, String actionName)
```

To samo co wyżej z tym, że dla każdego stanu.

```
public void addDefaultActionCellForEachState(Set<String> actionNames)
```

Usuwa akcje.

```
public void removeActions()
```

Zwraca ilość stanów.

```
public int size()
```

Wylicza na nowo prawdopodobieństwa w komórkach wiersza podanego stanu.

```
private void recalculateProbabilities(String state)
```

Zwraca akcję z największą wartością w wierszu określonym przez stan.

```
public String getActionWithMaxValue(String state)
```

Zwraca największą wartość we wierszu.

```
public double getMaxActionValue(String state)
```

Zwraca najlepszą akcję dla stanu.

```
public String getAction(String state)
```

Zwraca komórkę na przecięciu podanego stanu i akcji.

```
public ClusTableCell getCell(String state, String action)
```

Ustawia wartość komórki na przecięciu podanego stanu i akcji.

```
public void updateCellValue(String state, String action, Double value)
```

Ustawia ilość użyć komórki będącej na przecięciu podanego stanu i akcji.

```
public void updateCellUseCount(String state, String action, int value)
```

Zwraca aktualny stan. Używane dla konfiguracji globalnych, bo nie podajemy dla jakiego agenta.

```
public String getCurrentState(Map<String, Measure> measures)
```

Metoda abstrakcyjna do zaimplementowania przez stany globalne oraz holoniczne.

```
public abstract String getCurrentState(Map<String, Measure> measures,
                   AID aid);
```

Zwraca aktualny stan symulacji używając pakietu R, wywoływana jest odpowiednia metoda w zależności od tego, który mechanizm predykcji został podany w konfiguracji.

```
public String predictCurrentStateByR(Map<String, Double> measures, AID aid);
```

Zwraca aktualny stan symulacji w oparciu o środki klastrów.

```
public String predictCurrentStateByRCentres(double[] point,
                    String[] measureName, String[] clusterNames);
```

Zwraca aktualny stan symulacji w oparciu o drzewa decyzyjne.

```
public String predictCurrentStateByRTrees(double[] point,
                    String[] measureName, String[] clusterNames);
```

10. ClusTableGlobalStates.java

Implementacja ClusTableStates dla konfiguracji globalnej. Implementuje abstrakcyjną metodę, które zwraca aktualny stan systemu biorąc pod uwagę measurmenty globalne.

```
@Override
public String getCurrentState(Map<String, Measure> measures, AID aid) {
      Map<String, Double> currentMeasures = clusTableMeasures
                  .getCurrentMeasuresVector(measures);
      String currentState = predictCurrentStateByR(currentMeasures, aid);

      logger.info("Current global state: " + currentState);

      measurmentsHistory.add(currentMeasures);

      return currentState;
}
```

11. ClusTableHolonStates.java

Implementacja ClusTableStates dla konfiguracji holonicznej. Implementuje abstrakcyjną metodę, które zwraca aktualny stan systemu biorąc pod uwagę measurmenty lokalne.

```
@Override
public String getCurrentState(Map<String, Measure> measures, AID aid) {
      Map<String, Double> currentMeasures = clusTableMeasures
                  .getCurrentMeasuresVector(measures, aid);

      String currentState = predictCurrentStateByR(currentMeasures, aid);

      logger.info("Current holon state: " + currentState);

      measurmentsHistory.add(currentMeasures);

      return currentState;
}
```

12. RewardFunction.java

Reprezentuje funkcję nagrody. Obliczana dla akcji globalnych oraz holonicznych po sprawdzeniu jakie były rezultaty wykonania określonej akcji.

Reprezentuja funkcje wyciągnięte z funkcji nagrody.

```
private final Map<String, String> functions = new HashMap<String, String>();
```

Funkcja nagrody.

```
private final String rewardFunction;
```

Dla measurmentów, które mają w sobie funkcje agregujące – potrafi na podstawie stringowej reprezentacji zwrócić zagregowaną wartość.

```
protected AggregatorsManager aggregatorManager = new AggregatorsManager();
```

Inicjalizuje obiekt napisem oznaczającym funkcję nagrody.

```
public RewardFunction(String function) {
	rewardFunction = function;
	String parts[];
	for (String part : function.split("\\?")) {
		parts = part.split(";");
		if (parts.length == 1)
			functions.put(function, "true");
		else
			functions.put(parts[0].trim(), parts[1].trim());
	}
}
```

Zwraca funkcję nagrody.

```
public String getRewardFunction() {
	return rewardFunction;
}
```

Wstawia measurmenty w funkcję, musi być zaimplementowane osobno przez konfigurację holoniczną i globalną.

```
protected abstract String insertMeasures(String fun,
		Map<String, Measure> measures, AID aid);
```

Zwraca obliczoną funkcję nagrody .

```
public double getValue(Map<String, Double> parameters,
                  Map<String, Double> prevParameters, Map<String, Measure> measures,
                  AID aid)
```

Wstawia parametry do funkcji nagrody.

```
      private String insertParams(String str, Map<String, Double> params,
                  String suffix)
```

13. GlobalRewardFunction.java

Implementacja RewardFunction dla konfiguracji globalnej.

Wprowadza measurmenty w funkcję nagrody.

```
      @Override
      protected String insertMeasures(String fun, Map<String, Measure> measures,
                  AID aid) {
            aggregatorManager.setMeasures(measures);

            fun = aggregatorManager.insertAggregateValues(fun);

            aggregatorManager.aggregationFinished();
            return fun;

      }
```

14. HolonRewardFunction.java

Implementacja RewardFunction dla konfiguracji holonicznej.

Wprowadza measurmenty w funkcję nagrody.

```
      @Override
      protected String insertMeasures(String fun, Map<String, Measure> measures,
                  AID aid) {
            aggregatorManager.setMeasures(measures);
            fun = aggregatorManager.insertAggregateValues(fun);
            for (String name : measures.keySet()) {

                  try {
                        fun = fun.replace(name,
measures.get(name).getValues().get(aid)
                                    .toString());
                  } catch (NullPointerException e) {
                        fun = fun.replace(name, "0");
                  }
            }
            aggregatorManager.aggregationFinished();
            return fun;

      }
```

15. Helper.java

Klasa utilowa z publicznymi metodami statycznymi.

Zwraca parametry globalne konfiguracji takie jak holonsCount, dist, commissions, costOfCommission, timeFromCreationOfLastUnit.

```
public static Map<String, Double> getParameters(
            Map<AID, Schedule> oldSchedule, Map<AID, Schedule> newSchedule,
            SimInfo info, int timestamp)
```

Zwraca parametry lokalne konfiguracji takie jak holonDist, holonCommissions, holonCostOfCommissions.

```
public static Map<AID, Map<String, Double>> getHolonParameters(
            Map<AID, Schedule> oldSchedule, Map<AID, Schedule> newSchedule,
            SimInfo info, int timestamp)
```

16. Clustering.java

Główna klasa w pakiecie, która jest ładowana po podaniu jej w konfiguracji jako mlAlgorithm. Rozszerza MLAlogrithm. Odpowiedzialna za cały przebieg konfiguracji, od inicjalizacji po zapisanie nowej konfiguracji.

Pola klasy, głównie rzeczy zaczytana z konfiguracji.

```
private boolean learning;
private boolean useTrees;
private boolean overwriteConf;
private String minClusCount;
private String maxClusCount;
private boolean usePam;

private String schema;
private ClusTableGlobalMeasures globalMeasures;
private ClusTableHolonMeasures holonMeasures;
private ClusTableStates globalStates;
private ClusTableStates holonStates;
private final ClusTableActions<GlobalConfiguration> globalActions = new
ClusTableGlobalActions();
private final ClusTableActions<HolonConfiguration> holonActions = new
ClusTableHolonActions();

//historia pomiarów globalnych z przypisaniem do klastrów
private ClusTableObservations globalObservations = new ClusTableObservations();
//historia pomiarów holonicznych z przypisaniem do klastrów
private ClusTableObservations holonObservations = new ClusTableObservations();

private RewardFunction globalRewardFunction;
private double globalFactor;
private RewardFunction holonRewardFunction;
```

```
	private double holonsFactor;
	private boolean globalDeterministic = true;
	private boolean holonDeterministic = true;
	private Map<String, Double> defaultParams;

	List<Map<String, Measure>> measurmentsHistory = new LinkedList<Map<String,
Measure>>();

	private static final Logger log = Logger.getLogger(Clustering.class);
	private final ClusLogger logger = new ClusLogger();
```

Inicjalizuje obiekt. Inicjalizuje logger.

```
	public Clustering() {
		log.info("Clustering initialization");
		System.out
				.println("---------------------------------Clustering ini-
tialization");
		logger.init();
	}
```

Inicjalizuje stany oraz akcję. Tworzy komórki itp.

```
	public void init(ClusTableStates states, ClusTableActions<?> actions) {
		if (states == null || actions == null)
			return;
		ClusTableCell cell;
		List<ClusTableCell> cells;
		for (String name : states.getValues().keySet()) {
			cells = new ArrayList<ClusTableCell>();
			for (String actionName : actions.getActions().keySet()) {
				cell = new ClusTableCell();
				cell.setState(name);
				cell.setAction(actionName);
				cell.setValue(0.0);
				cells.add(cell);
			}
			states.getRows().put(name, cells);
		}
	}
```

Zwraca konfigurację globalną systemu.

```
	@Override
	public GlobalConfiguration getGlobalConfiguration(
			Map<AID, Schedule> oldSchedules, Map<AID, Schedule> newSchedules,
			SimInfo info, boolean exploration, AlgorithmAgentParent agent)
```

Dla trybu eksploracji woła inną funkcję, dla trybu bez eksploracji wylicza measurmenty, wyznacza stan, a następnie dla tego stanu wybiera najlepszą akcję.

```
		log.info("Getting global conf, learning mode enabled ? " + isLearning());

		if (exploration)
			return getGlobalConfiguration(oldSchedules, newSchedules, info,
					agent);
		else {
			if (globalStates == null)
				return null;
```

```
                Map<String, Measure> measures = calculateMeasures(oldSchedules,
                        newSchedules, agent);
                String currentState = globalStates.getCurrentState(measures);
                String action = globalStates.getActionWithMaxValue(currentState);
                return globalActions.getActions().get(action);

        }
```

Wyznacza globalną konfigurację oraz przeprowadza eksplorację.

```
private GlobalConfiguration getGlobalConfiguration(
                Map<AID, Schedule> oldSchedules, Map<AID, Schedule> newSchedules,
                SimInfo info, AlgorithmAgentParent agent)
```

Wylicza measurmenty, wyznacza stan, a następnie dla tego stanu wybiera najlepszą akcję.

```
        if (globalStates == null)
                return null;
        Map<String, Measure> measures = calculateMeasures(oldSchedules,
                        newSchedules, agent);
        String currentState = globalStates.getCurrentState(measures);
        String action = globalStates.getAction(currentState);
```

Jeśli to pierwszy przebieg to nie przeprowadza eksploracji, tylko pobiera parametry.

```
if (previousCell == null) {
                previousCell = globalStates.getCell(currentState, action);
                // globalStates.updateCellUseCount(currentState, action,
                // previousCell.getUseCount() + 1);
                previousParameters = Helper.getParameters(oldSchedules,
                                newSchedules, info, timestamp);
                return globalActions.getActions().get(action);

        }
```

Jeśli już były jakieś przebiegi to aktualizuje wartości komórek dla wybranego stanu.

```
        Map<String, Double> parameters = Helper.getParameters(oldSchedules,
                        newSchedules, info, timestamp);

        double value = calculateCellValue(globalStates, globalRewardFunction,
                        parameters, previousParameters, measures, null, globalFactor,
                        currentState, previousCell, globalDeterministic);

        globalStates.updateCellValue(previousCell.getState(),
                        previousCell.getAction(), value);
        globalStates.updateCellUseCount(previousCell.getState(),
                        previousCell.getAction(), previousCell.getUseCount() + 1);

        previousCell = globalStates.getCell(currentState, action);
        previousParameters = parameters;


        return globalActions.getActions().get(action);
```

Zwraca konfigurację dla holonów.

```
@Override
public Map<AID, HolonConfiguration> getHolonsConfiguration(
            Map<AID, Schedule> oldSchedules, Map<AID, Schedule> newSchedules,
            SimInfo info, boolean exploration, AlgorithmAgentParent agent)
```

Dla trybu eksploracji woła inną funkcję, dla trybu bez eksploracji wylicza measurmenty, wyznacza stan, a następnie dla tego stanu wybiera najlepszą akcję.

```
log.info("Getting holon conf, learning mode enabled ? " + isLearning());

        if (exploration)
            return getHolonsConfiguration(oldSchedules, newSchedules, info,
                    agent);
        else {
            if (holonStates == null)
                return null;
            Map<AID, HolonConfiguration> configurations = new HashMap<AID,
HolonConfiguration>();
            Map<String, Measure> measures = calculateMeasures(oldSchedules,
                    newSchedules, agent);
            Set<AID> aids;
            if (newSchedules != null && newSchedules.size() > 0)
                aids = newSchedules.keySet();
            else
                aids = oldSchedules.keySet();

            for (AID holon : aids) {
                String currentState = holonStates.getCurrentState(measures,
                        holon);
                String action = holonStates.getActionWithMaxValue(current-
State);
                configurations
                        .put(holon, holonActions.getActions().get(ac-
tion));

            }
            return configurations;

        }
```

Zwraca konfigurację holonów dla trybu z eksploracją.

```
private Map<AID, HolonConfiguration> getHolonsConfiguration(
            Map<AID, Schedule> oldSchedules, Map<AID, Schedule> newSchedules,
            SimInfo info, AlgorithmAgentParent agent)
```

Najpierw oblicza measurmenty oraz inicjalizuje obliczenia.

```
        if (holonStates == null)
            return null;
        Map<AID, HolonConfiguration> configurations = new HashMap<AID, Holon-
Configuration>();
        Map<String, Measure> measures = calculateMeasures(oldSchedules,
                newSchedules, agent);
        ClusTableCell previousHolonCell;
```

```
Map<AID, Map<String, Double>> holonParams = Helper.getHolonParameters(
                oldSchedules, newSchedules, info, timestamp);
Set<AID> aids;
if (newSchedules != null && newSchedules.size() > 0)
      aids = newSchedules.keySet();
else
      aids = oldSchedules.keySet();
```

Następnie dla każdego holonu oblicza wartości komórek, aktualizuje je oraz aktualizuje liczbę użyć.

```
                String currentState = holonStates.getCurrentState(measures, holon);
                String action = holonStates.getAction(currentState);
                previousHolonCell = previousCells.get(holon);
                if (previousHolonCell == null) {
                      previousHolonCell = holonStates.getCell(currentState,
action);
                      previousCells.put(holon, previousHolonCell);
                      holonStates.updateCellUseCount(currentState, action,
                                  previousHolonCell.getUseCount() + 1);
                      configurations
                                  .put(holon, holonActions.getActions().get(ac-
tion));
                      previousHolonParameters.put(holon, holonParams.get(holon));
                      continue;
                }
                Map<String, Double> parameters = holonParams.get(holon);

                double value = calculateCellValue(holonStates, holonRewardFunction,
                            parameters, previousHolonParameters.get(holon), meas-
ures,
                            holon, holonsFactor, currentState, previousHolonCell,
                            holonDeterministic);

                holonStates.updateCellValue(previousHolonCell.getState(),
                            previousHolonCell.getAction(), value);
                holonStates.updateCellUseCount(previousHolonCell.getState(),
                            previousHolonCell.getAction(),
                            previousHolonCell.getUseCount() + 1);

                previousHolonCell = holonStates.getCell(currentState, action);
                previousCells.put(holon, previousHolonCell);
                previousHolonParameters.put(holon, parameters);


                configurations.put(holon, holonActions.getActions().get(action));
```

Ustawia parametry algorytmu podane w konfiguracji.

```
      @Override
      public void setAlgorithmParameters(Map<String, String> parameters)
```

Oblicza wartość komórki – na podstawie funkcji nagrody.

```
 private double calculateCellValue(ClusTableStates states,
                RewardFunction function, Map<String, Double> parameters,
```

```
                Map<String, Double> prevParameters, Map<String, Measure> measures,
                AID aid, double factor, String currentState,
                ClusTableCell previousCell, boolean deterministic)
```

Wykonywane na końcu symulacji – zapisuje do pliku aktualną konfigurajcję.

```
@Override
      public void save(String clusTableFileName, String saveFileName)
                  throws Exception
```

Wywoływana przy starcie. Inicjalizuje obiekt poprzez wczytanie konfiguracji oraz stworzenie struktur używanych przez R do predykcji stanu.

```
      @Override
      public void init(String fileName) {
            try {
                  ClusTableStructureParser.parse(fileName, this);
                  log.info("Init cluse table, useTrees: " + this.isUseTrees());

                  if (this.isUseTrees()) {
                        initTrees();
                  } else {
                        initCentres();
                  }

            } catch (ParseException e) {
                  e.printStackTrace();
                  System.exit(0);
            }

      }
```

Najważniejsza metoda w tej klasie. Jest ona wykonywana, gdy włączony jest clustering. Służy do generacji klastrów oraz zapisu ich w konfiguracji.

```
public void clustering()
```

Configuracja globalna :

Na początku pobierane są measurmenty, a na ich podstawie tworzony jest matrix, który posłuży do wyliczania stanów systemu (klastrów).

```
List<Map<String, Double>> globalHistory = globalStates
                        .getMeasurmentsHistory();
            List<String> rNames = new LinkedList<String>();
            List<String> cNames = new LinkedList<String>();

            // fill the names
            for (String mn : globalHistory.get(0).keySet()) {
                  cNames.add(mn);
            }

            double[] matrixValues = new double[globalHistory.size() * cNames.size()];

            Integer nr = 0;
            int measureNr = 0;
            for (Map<String, Double> values : globalHistory) {
                  rNames.add((nr++).toString());
```

```
            for (String measure : values.keySet()) {
                  matrixValues[measureNr++] = values.get(measure);
            }
      }

      RUtils rutils = new RUtils();
      rutils.start();

      log.info("Global history:\n" + globalHistory);
      log.info("Global rNames:\n" + rNames);
      log.info("Global cNames:\n" + cNames);
```

Następnie przy użyciu algorytmu pamk (odmiana kmeasn) obliczany jest clustering. Zwracany jest obiekt kmeans, który następnie jest zamieniany na środki klastrów.

```
      REXP kmeans = rutils.kmeans(matrixValues,
                  rNames.toArray(new String[] {}),
                  cNames.toArray(new String[] {}));

      List<Map<String, Double>> centres = rutils.getCentres(kmeans,
                  cNames.toArray(new String[] {}));
```

Następnie dotychczasowe stany są usuwane, na podstawie klastrów tworzone są nowe, wypełniaone zostaną wiersze dla każdego stanu.

```
            globalObservations.clean();
            addObservations(globalObservations, kmeans, cNames, sNames);


// remove learning states
      globalStates.removeStates();

      // add generated clusters

      for (int s = 0; s < centres.size(); s++) {
            globalStates.addState(statePrefix + s, centres.get(s));
      }

      // remove all action cells
      globalStates.removeActions();

      // add actions with default values
      globalStates.addDefaultActionCellForEachState(globalActions
                  .getActions().keySet());
```

Dla konfiguracji holonicznej sprawa wygląda podobnie.

Wyliczany jest matrix wartości, który następnie będzie wejściem clusteringu.

```
List<Map<String, Double>> holonHistory = holonStates
                  .getMeasurmentsHistory();

      rNames = new LinkedList<String>();
      cNames = new LinkedList<String>();

      // fill the names
      for (String mn : holonHistory.get(0).keySet()) {
            cNames.add(mn);
      }
```

```
matrixValues = new double[holonHistory.size() * cNames.size()];

nr = 0;
measureNr = 0;
for (Map<String, Double> values : holonHistory) {
	rNames.add((nr++).toString());

	for (String measure : values.keySet()) {
		matrixValues[measureNr++] = values.get(measure);
	}
}

log.info("Holon history:\n" + holonHistory);
log.info("Holon rNames:\n" + rNames);

log.info("Holon cNames:\n" + cNames);
```

Następnie uruchamiany jest pamk z klasy RUtils oraz obliczane są klastry.

```
kmeans = rutils.kmeans(matrixValues, rNames.toArray(new String[] {}),
		cNames.toArray(new String[] {}));

centres = rutils.getCentres(kmeans, cNames.toArray(new String[] {}));


rutils.end();
```

Po tym usuwane są stare stany oraz generowane są nowe. Tabela jest wypełniana domyślnymi wartościami (zerami).

```
// remove learning states
holonStates.removeStates();

// add generated clusters

for (int s = 0; s < centres.size(); s++) {
	holonStates.addState(statePrefix + s, centres.get(s));
}

	holonObservations.clean();
	addObservations(holonObservations, kmeans, cNames, sNames);


// remove all action cells
holonStates.removeActions();

// add actions with default values
holonStates.addDefaultActionCellForEachState(holonActions.getActions()
		.keySet());

log.info("Clustering completed...");
```

17. RUtils.java

Klasa implementuja komunikację z silnikiem R. Wykonuje funkcje klastrujące z R.

Dodatkowe pakiety, które muszą być załadowany dla poprawnego działania systemu.

```
public static final String[] REQUIRED_PACKAGES = { "fpc", "clue", "rpart" };
```

Silnik z rJava.

```
	private static Rengine rengine = null;
```

Uruchamia silnik R oraz ładuje niezbędne biblioteki do R.

```
public synchronized Rengine start()
```

Oblicza kmeans. Values – matrix wartości, rnames – nazwy wierszy, cnames – nazwy kolumn

```
public REXP kmeans(double[] values, String[] rnames, String[] cnames)
{
		// oblicza dolne i gorne ograniczenie na ilosc klastrow
		// z wyrazen podanych w konfiguracji
		minClusCountExpr = minClusCountExpr.replaceAll("N",
				String.valueOf(rnames.length));
		maxClusCountExpr = maxClusCountExpr.replaceAll("N",
				String.valueOf(rnames.length));

		double minClusCount = Calculator.calculate(minClusCountExpr);
		double maxClusCount = Calculator.calculate(maxClusCountExpr);

		logger.info("MIN clusters count: " + minClusCount
				+ " MAX clusters count: " + maxClusCount);

		// przypisuje lokalne zmienne do zmiennych srodowiska R

		logger.info("values:\n" + Arrays.toString(values));
		rengine.assign(CELLS_NAME, values);
		rengine.assign(ROWS_NAME, rnames);
		rengine.assign(COLS_NAME, cnames);

		String matrixCmd = "x <- matrix(" + CELLS_NAME + ", nrow="
				+ rnames.length + ", ncol=" + cnames.length
				+ ", byrow=TRUE, dimnames=list(" + ROWS_NAME + "," +
COLS_NAME
				+ "))";
		logger.info("Creating matrix command: " + matrixCmd);

		REXP result = rengine.eval(matrixCmd);

		//przeprowadza klastering
		String kmeansCmd = "km <- pamk(x," + ((int) minClusCount) + ":"
				+ ((int) maxClusCount) + ",usepam="
				+ String.valueOf(usePam).toUpperCase() + ")";
```

```
        Logger.info("Kmeans command: " + kmeansCmd);
        REXP km = rengine.eval(kmeansCmd);

        // rengine.eval("print(km$nc)");

        Logger.info("Optimal number of clusters : "
                    + ((REXP) km.asVector().get(1)).asInt());

        return km;

    }
```

Dokonuje predykcji aktualnego stanu na podstawie środków klastrów.

```
    public String predictStateByCentres(double[] point, String[] measureName,
            String[] clusterNames, String centresStructureName) {
        String cmd = "";
        final String pointMatrixName = "pmatrix";
        final String newPointName = "newPoint";

        // first assing point as matrix
        rengine.assign(newPointName, point);
        cmd = pointMatrixName + " <- matrix(" + newPointName + ",nrow=1,ncol="
                    + point.length + ")";
        System.out.println("Matrix from new point cmd: " + cmd);
        rengine.eval(cmd);

        rengine.eval("print(pmatrix)");

        // predict cluster number
        cmd = "cl_predict(" + centresStructureName + "," + pointMatrixName
                    + ")";
        System.out.println("Predict cmd: " + cmd);
        REXP predict = rengine.eval(cmd);

        System.out.println("Cluster index: " + predict.asInt());

        return clusterNames[predict.asInt() - 1];

    }
```

Dokonuje predykcji stanu na podstawie drzew decyzyjnych.

```
    public String predictStateByTree(double[] point, String[] measureNames,
            String[] clusterNames, String treeStructureName) {

        String cmd = null;
        final String testDataName = "testData";
        final String toPredict = "toPredict";
        final String toPredictDataFrame = "toPredictDataFrame";

        int nrow = 1;
        int ncol = point.length;

        rengine.assign(COLS_NAME, measureNames);

        rengine.assign(testDataName, point);

        cmd = toPredict + " <- matrix(" + testDataName + ", nrow=" + nrow
                    + ", ncol=" + ncol + ", byrow=TRUE, dimnames=list(1:" + nrow
```

```
                        + "," + COLS_NAME + "))";

                System.out.println("Test data matrix cmd: " + cmd);
                rengine.eval(cmd);

                cmd = toPredictDataFrame + " <- data.frame(" + toPredict + ")";
                rengine.eval(cmd);

                final String treePredict = "treePredict";

                cmd = treePredict + " <- predict(" + treeStructureName + ","
                                + toPredictDataFrame + ",type=\"vector\")";
                System.out.print("Predict command: " + cmd);
                REXP predictionResult = rengine.eval(cmd);

                rengine.eval("print(" + treePredict + ")");

                int predictedState = (int) (predictionResult.asDoubleArray()[0]);

                System.out.println("Tree predict result: "
                                + clusterNames[predictedState]);

                return clusterNames[predictedState];

        }
```

Kończy działanie R.

```
        public void end() {
                if (rengine != null) {
                        rengine.end();
                }
        }
```

Klasa dzięki, której cały output z R trafia na console.

```
class TextConsole implements RmainLoopCallbacks
```

8. Testy rozwiązania

Przeprowadzone testy miały na celu sprawdzenie poprawności działania zaimplementowanego modułu uczenia maszynowego, sprawdzenie jego wydajności oraz porównanie do wcześniejszego rozwiązania pod względem jakości.

8.1 Realizacja

Przyjęto następujące zasady:

- Tryb exploration == true
- Zostały wykorzystane benchmarki Panów H. Li oraz A. Lim – publicznie dostępne zestawy do porównywania jakości rozwiązań problemów PDPTW
- Wszystkie testy z mieszanym rozmieszczeniem zamówień (RC)
- W każdym przypadku test był uruchamiany z wąskimi oraz szerokimi oknami czasowymi
- Testy dla wersji statycznej oraz dynamicznej
- Tryb uczenia był włączany: po zmianie rozmiaru problemu, po zmianie rozmiaru okien czasowych, po zmianie static/dynamic
- Tryb uczenia obejmował: jedno uruchomienie w trybie uczenia – generacja stanów, wyzerowanie tabeli oraz jedno uruchomienie w trybie pracy – włączona eksploracja, wstępne uzupełnienie tabeli
- Dla tego samego typu benchmarku, tryb uczenia włączony był tylko raz, na początku, czyli testy zaprezentowane na pojedynczym wykresie obejmowały uruchomienie jeden raz trybu uczenia na pierwszym zestawie

- Podczas testów dla konfiguracji globalnej użyto następującej funkcji nagrody : *2*_holonsCount/holonsCount+bestDist/dist+_costOfCommission/costOfCommission*. Wartości z podkreśleniem odpowiadają wartości parametru z poprzednich obliczeń. Do obliczeń przyjęto stały parametr bestDist o wartości 828.94, który był używany wcześniej przy użyciu QLearningu. Podczas testów nie zmienialiśmy funkcji nagrody, była ona taka sama dla QLearningu jak również dla Clusteringu.
- Dla konfiguracji lokalnej (holonicznej) użyto natomiast funkcji : *2*_holonCommissions/holonCommissions+_costOfCommission/costOfCommission*. Wartości z podkreśleniem odpowiadają wartości parametru z poprzednich obliczeń. Podczas testów nie zmienialiśmy funkcji nagrody, była ona taka sama dla QLearningu jak również dla Clusteringu.

8.2 Wyniki

Nazewnictwo benchmarków: rodzaj_problemu[static,dynamic], wielkość_problemu[100..1000],szerokość_okien_czasowych[wąskie,szerokie].
Kolejne słupki na wykresie przedstawiają wyniki dla kolejnego zestawu zleceń z tego samego typu benchmarku np. 1 oznacza, że jest to zestaw 1.
W pierwszym słupku znajdują się wyniki dla QLearningu – rozwiązania istniejącego w systemie zanim zaczęliśmy nad nim pracować. W dwóch kolejnych słupkach znajdują się wyniki dla Clusteringu. W drugim słupku są wyniki dla użycia cl_predict jako funkcji predykcji stanu, a w trzecim drzew decyzyjnych. W dwóch przypadkach dodatkowo przeprowadzono testy w sytuacji gdy zmianę konfiguracji mamy włączoną tylko dla konfiguracji globalnej – 5 kolumna.
Dodatkowo należy zwrócić uwagę na porównanie czasu wykonania. Testy przeprowadzaliśmy

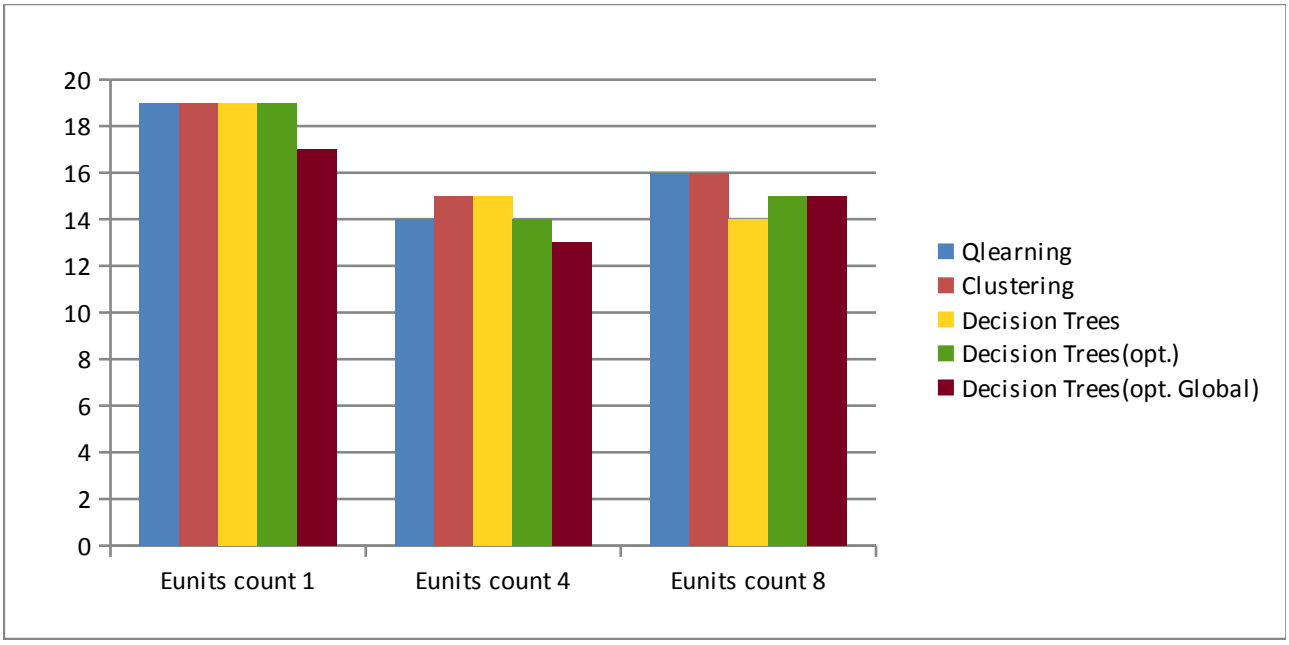
na trzech komputerach (A,B,C). Przy czym C miał największą moc obliczeniową, a A najmniejszą. Testy porównawcze dla QLearningu i Clusteringu były przeprowadzane zawsze na tym samym komputerze (A lub C), natomiast testy dla drzew decyzyjnych były przeprowadzone na komputerze B. Stało się tak ponieważ podczas wykonywania testów dla drzew decyzyjnych nie byliśmy w posiadaniu komputera C, a komputer A był bardzo wolny, więc aby przyśpieszyć badania zdecydowaliśmy się na użycie komputera B. Dlatego też wyniki porównania czasu symulacji dla drzew decyzyjnych mogą nie być miarodajne.

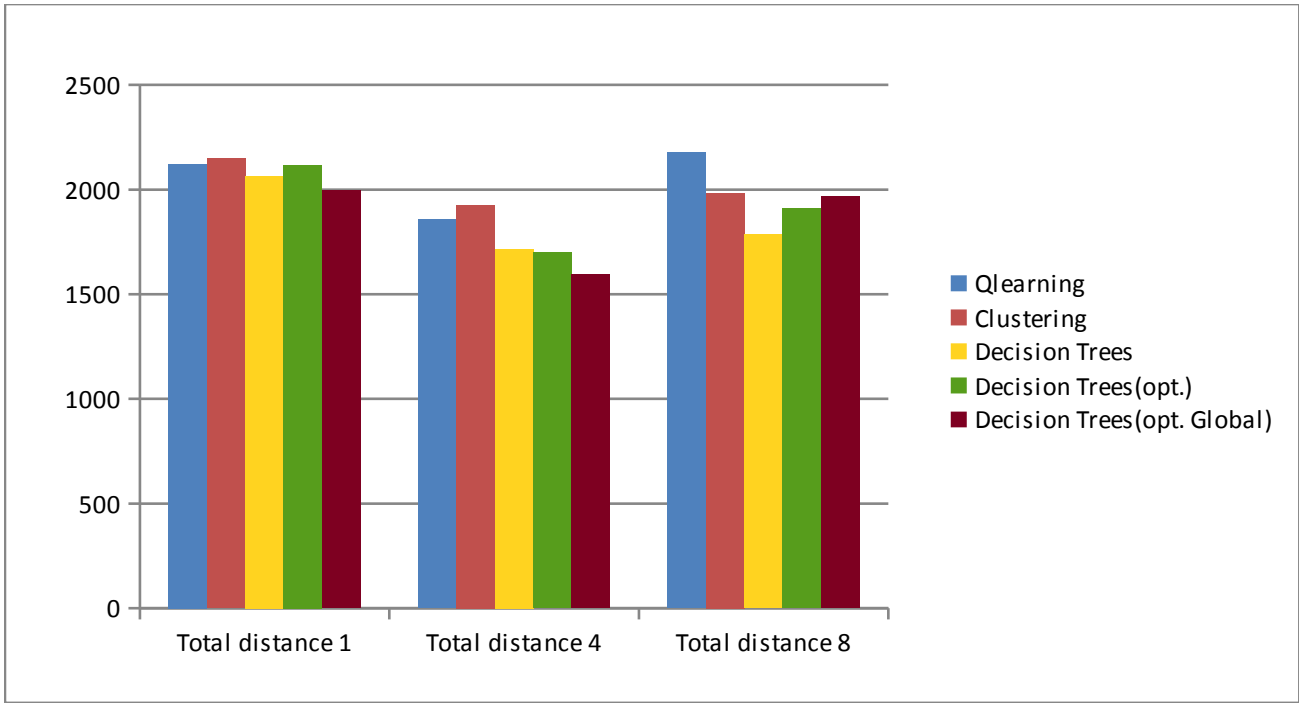
Pliki z wynikami :
Pliki dla poniższych testów znajdują się w folderze testy. W środku znajduje się 5 folderów :

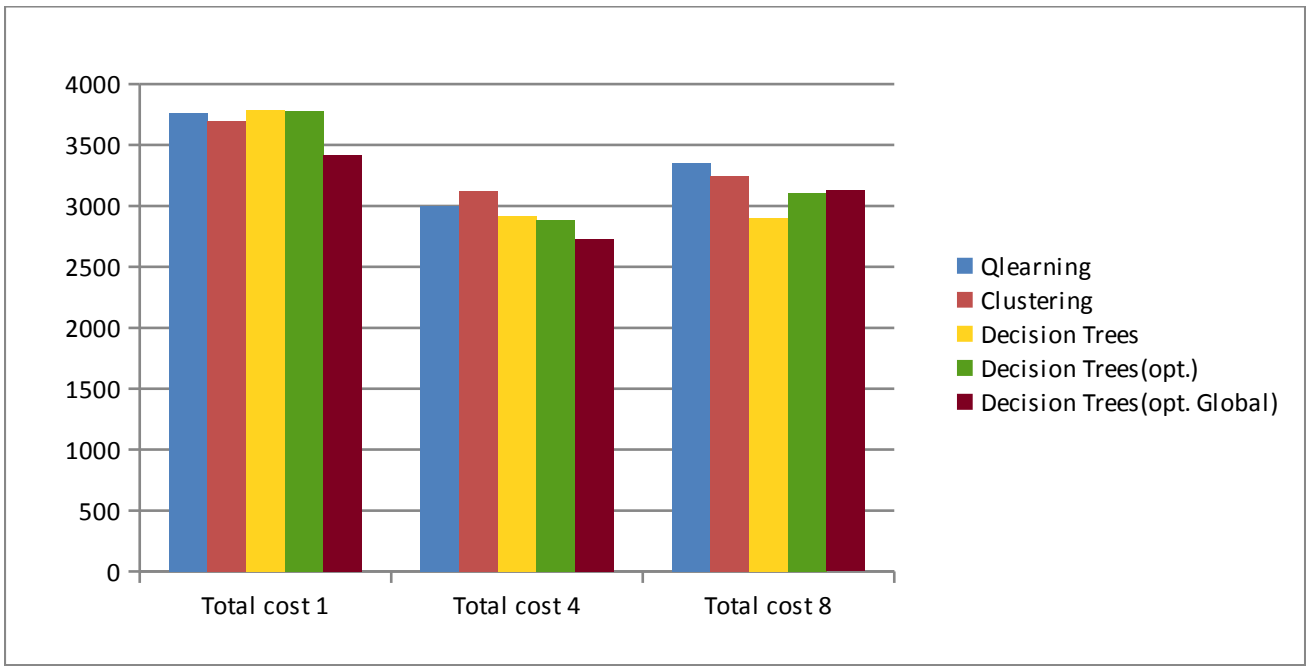
- qlearn – testy dla Q-Learningu
- clust – wyniki dla Clusteringu
- decision trees – wyniki dla drzew decyzyjnych (stare miary)
- decision trees – opt – wyniki dla drzew decyzyjnych (wybrane miary)
- decision trees – opt -global only – wyniki dla drzew decyzyjnych (wybrane miary, wyłączona zmiana konfiguracji dla holonów)

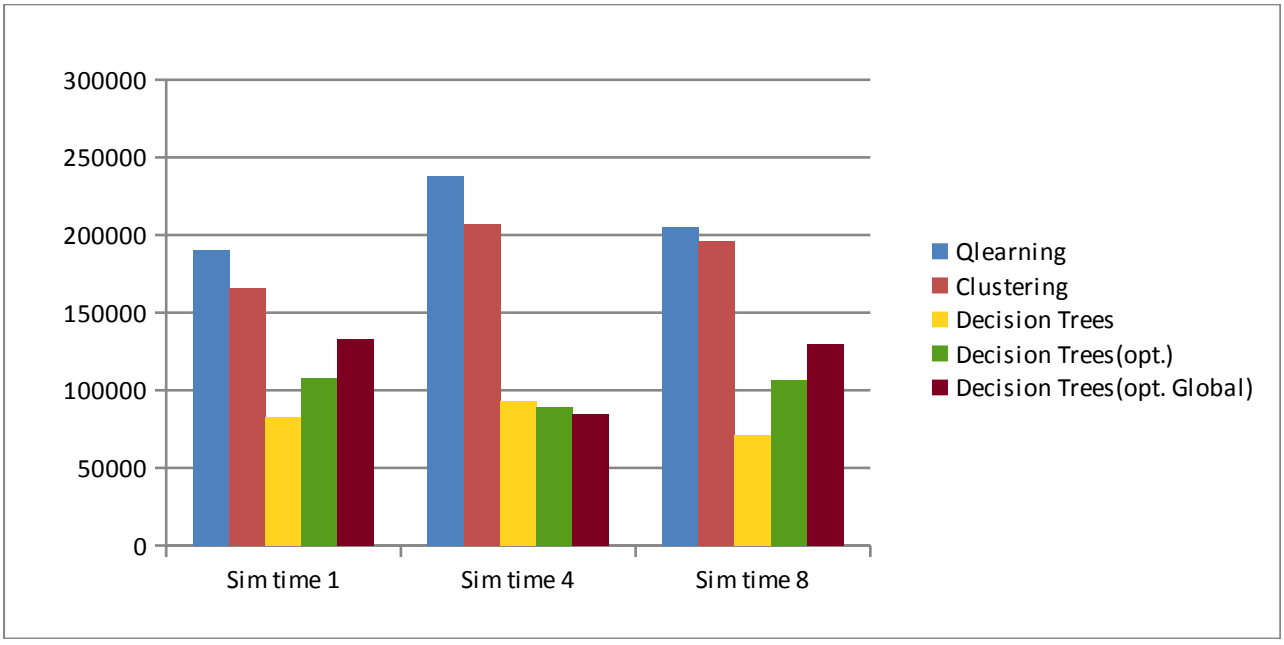
W każdym z powyższych folderów znajdują się dwa foldery „static” - wyniki dla problemu statycznego oraz „dynamic” - wyniki dla problemu dynamicznego. We wnętrzu tych folderów znajdują się foldery, których nazwa oznacza rozmiar problemu, np. 100. W środku znajdują się pliki pliki xls z wynikami testów, nazwy takie same jak nazwy odpowiadających im benchmarków.

1. static, 100, wąskie (lrc101, lrc104, lrc108)



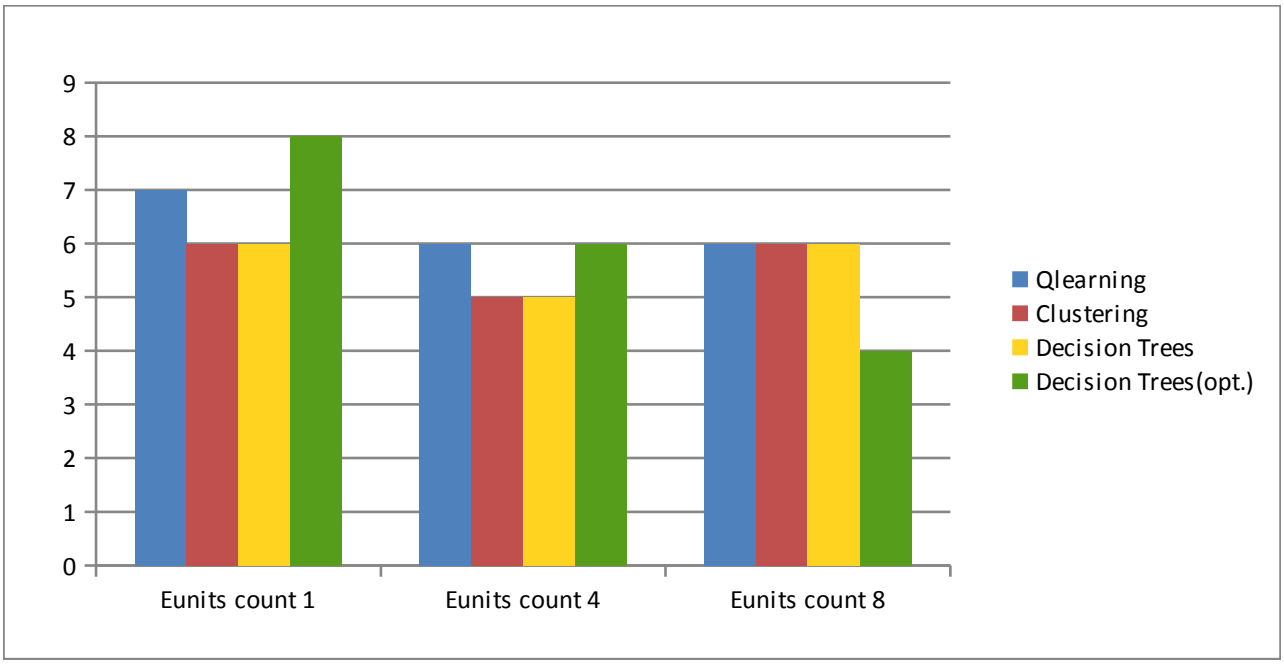

2500
2000
1500
1000
500
0
Total distance 1
Total distance 4
Total distance 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)
Decision Trees(opt. Global)

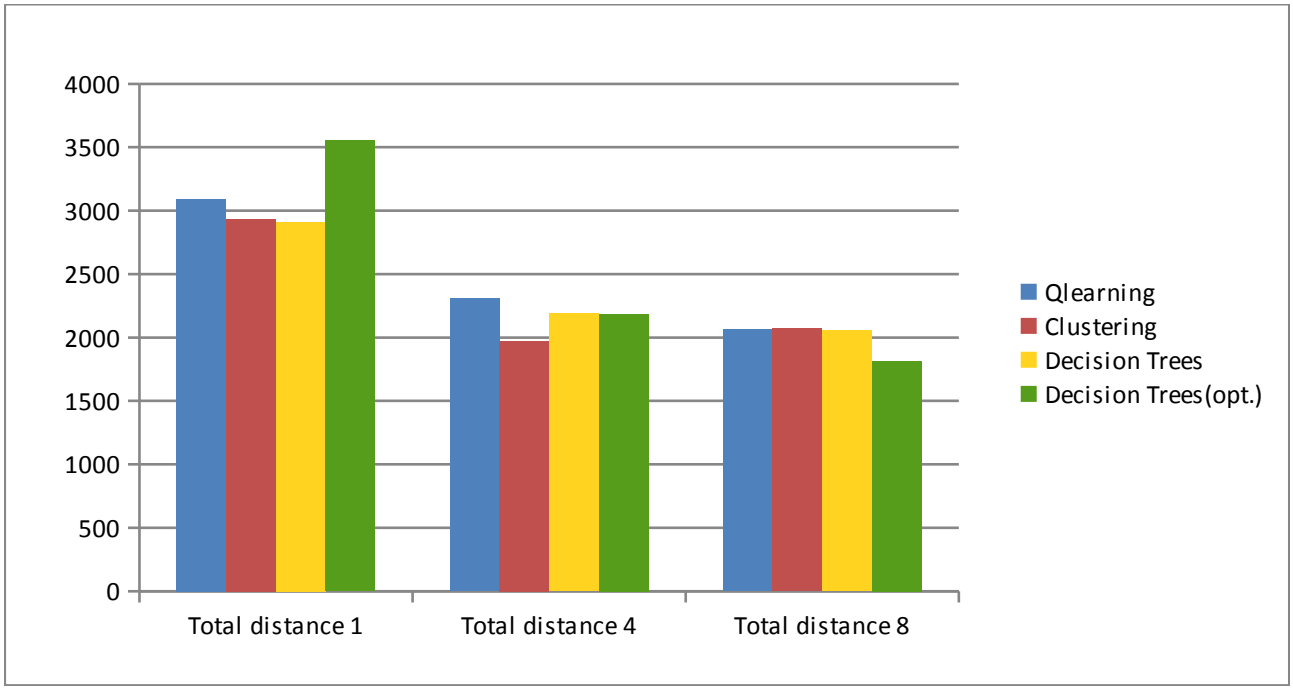

4000
3500
3000
2500
2000
1500
1000
500
0
Total cost 1
Total cost 4
Total cost 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)
Decision Trees(opt. Global)

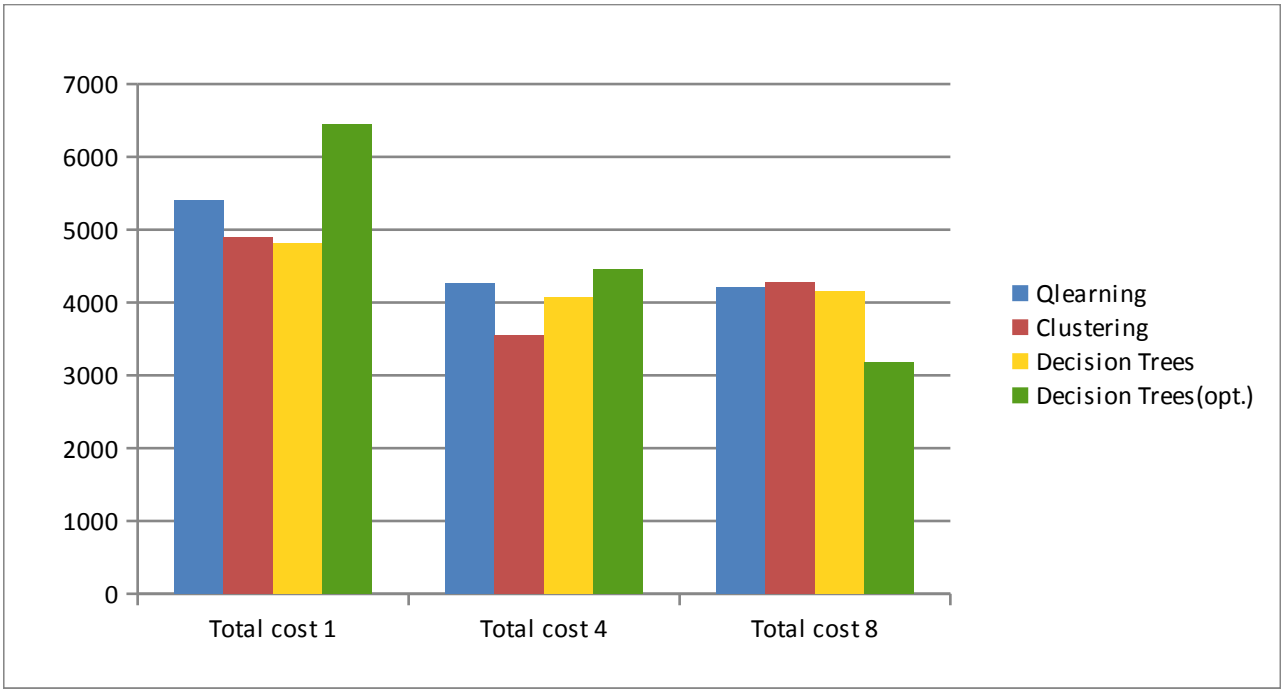


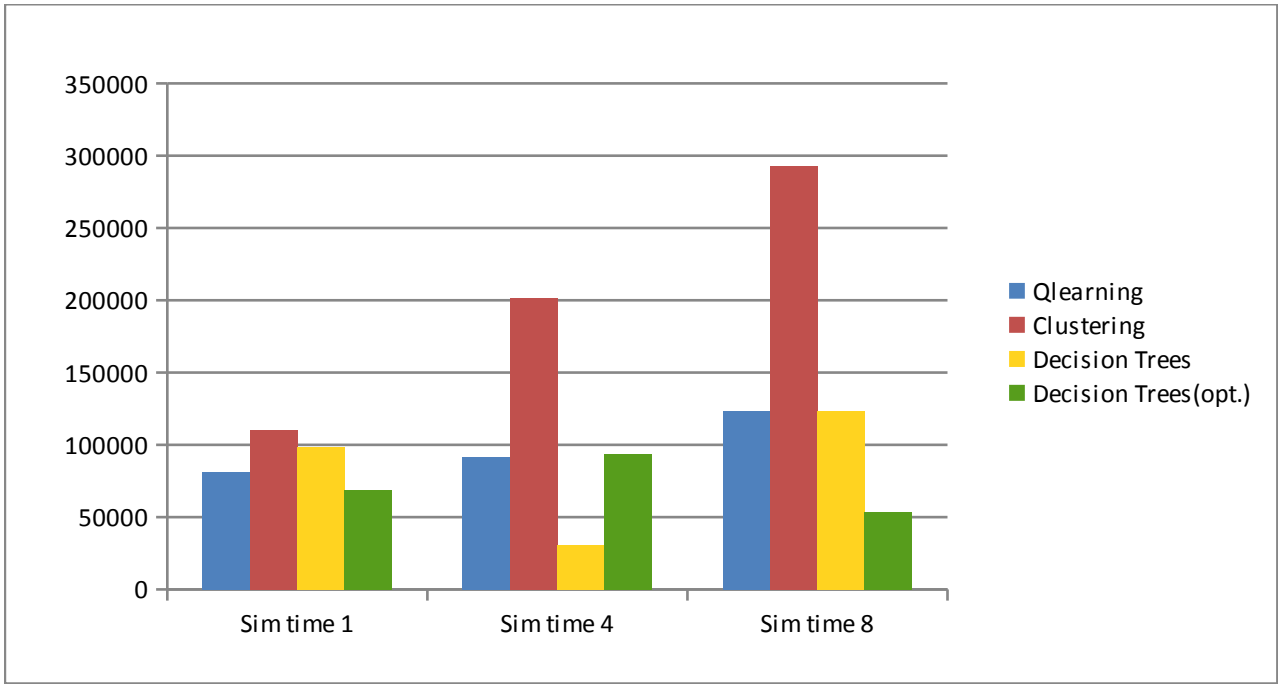
Algorytm Clusteringu dla początkowych pomiarów (lrc101, lrc104) dał gorsze wyniki od QLearningu, jednak w ostatnim teście (lrc108) uzyskaliśmy o wiele lepsze wyniki co może świadczyć o tym, że nasz system szybciej i lepiej się uczy. W wszystkich przypadkach czas trwania symulacji był krótszy. W przypadku drzew decyzyjnych uzyskaliśmy jeszcze lepsze rezultaty. Użycie optymalnych miar nie miało w tym wypadku dużego znaczenia. Wyłączenie zmian konfiguracji dla holonów było dobrym pomysłem i dało nieco lepsze wyniki.

2. static, 100, szerokie (lrc201, lrc204, lrc208)



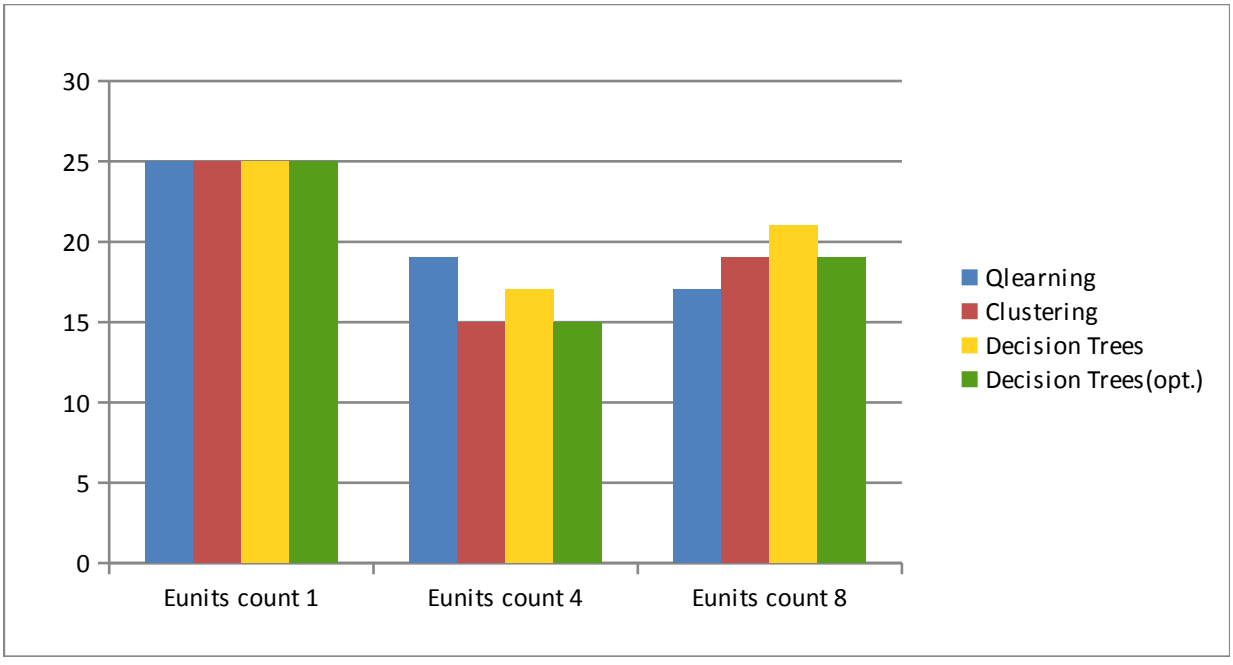

4000
3500
3000
2500
2000
1500
1000
500
0
Total distance 1
Total distance 4
Total distance 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)

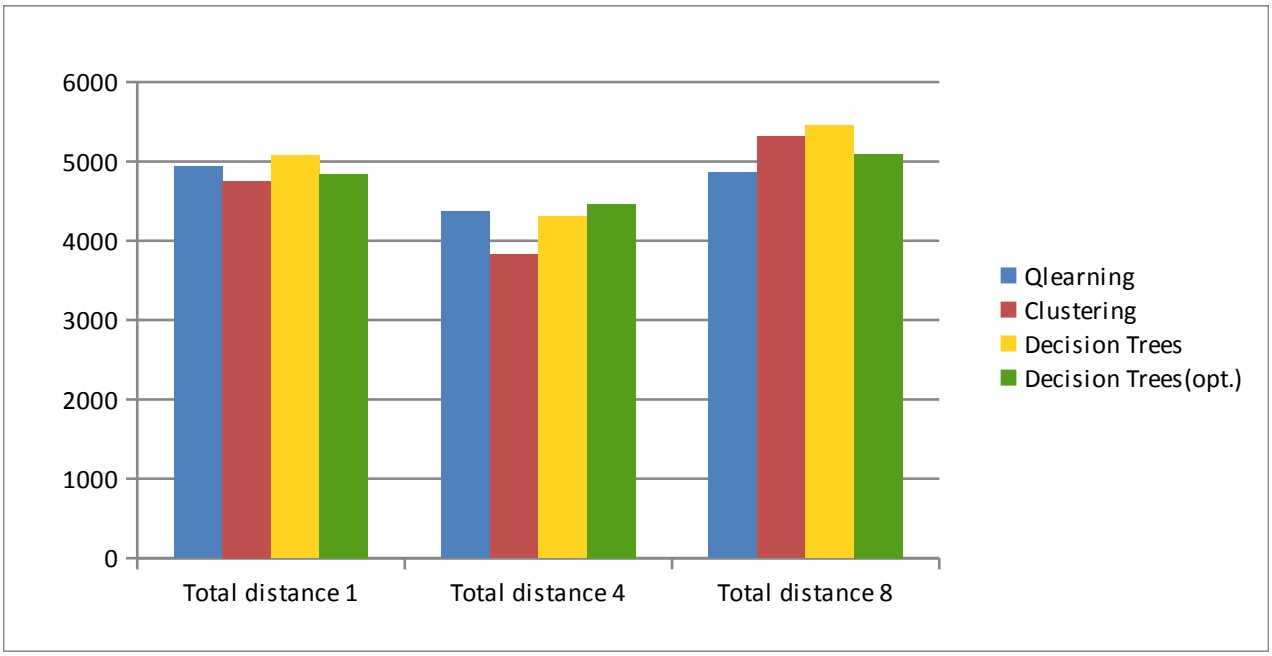

7000
6000
5000
4000
3000
2000
1000
0
Total cost 1
Total cost 4
Total cost 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)

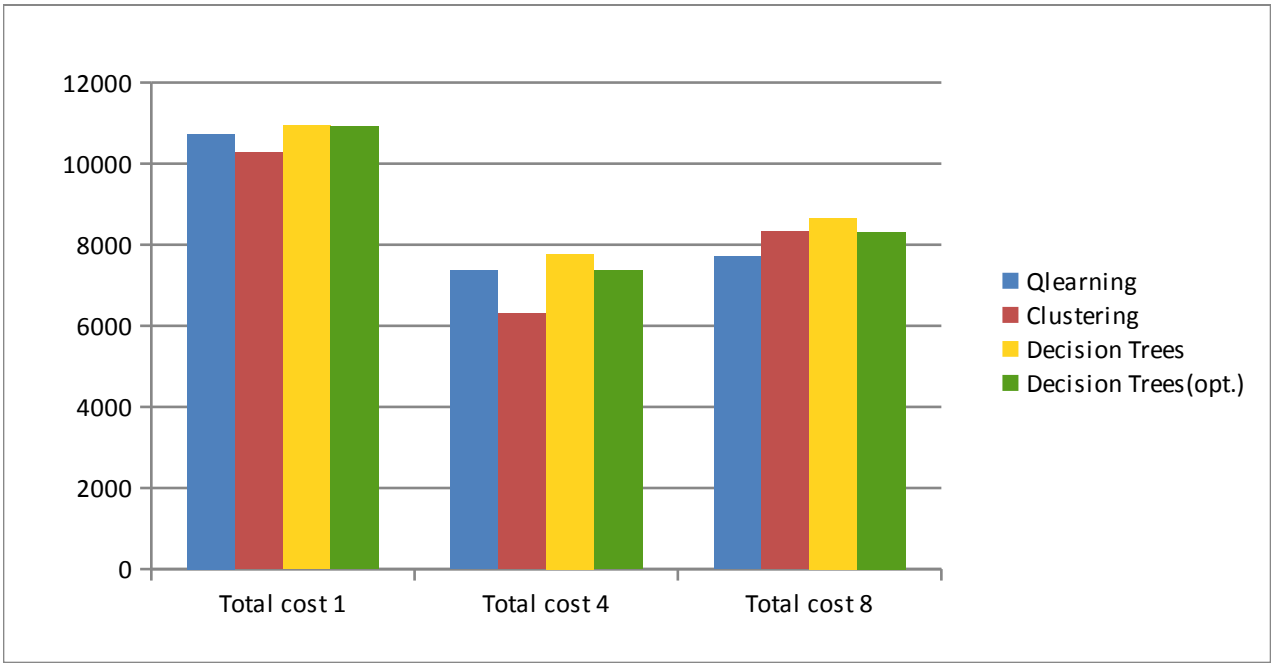


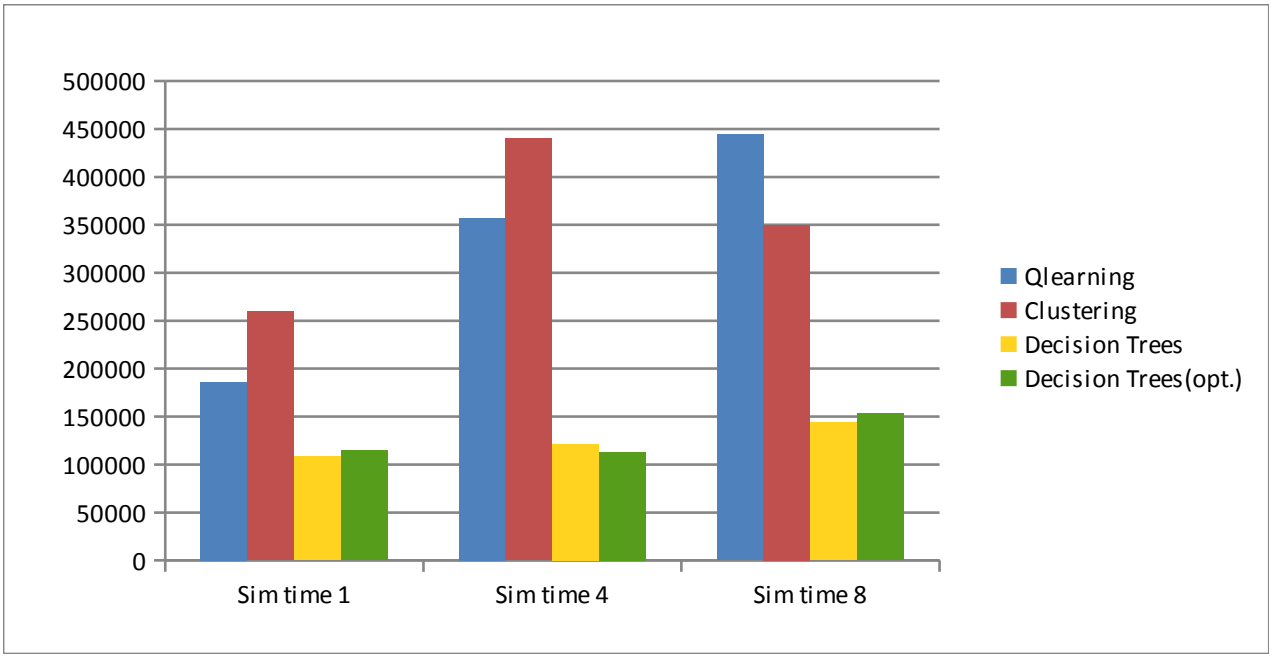
W tym przypadku algorytm Clusteringu znacznie przeważa nad QLearningiem, zostało to niestety okupione o wiele dłuższym czasem symulacji. Podobnie jest z drzewami decyzyjnymi, w każdym wypadku dały one lepszy rezultat niż Qlearning. Użycie optymalnych miar dało świetne wyniki dla 8 zestawu testów, może to wynikać z faktów, że ten benchmark był brany pod uwagę przy optymalizacji.

3. static, 200, wąskie (lrc121, lrc124, lrc128)



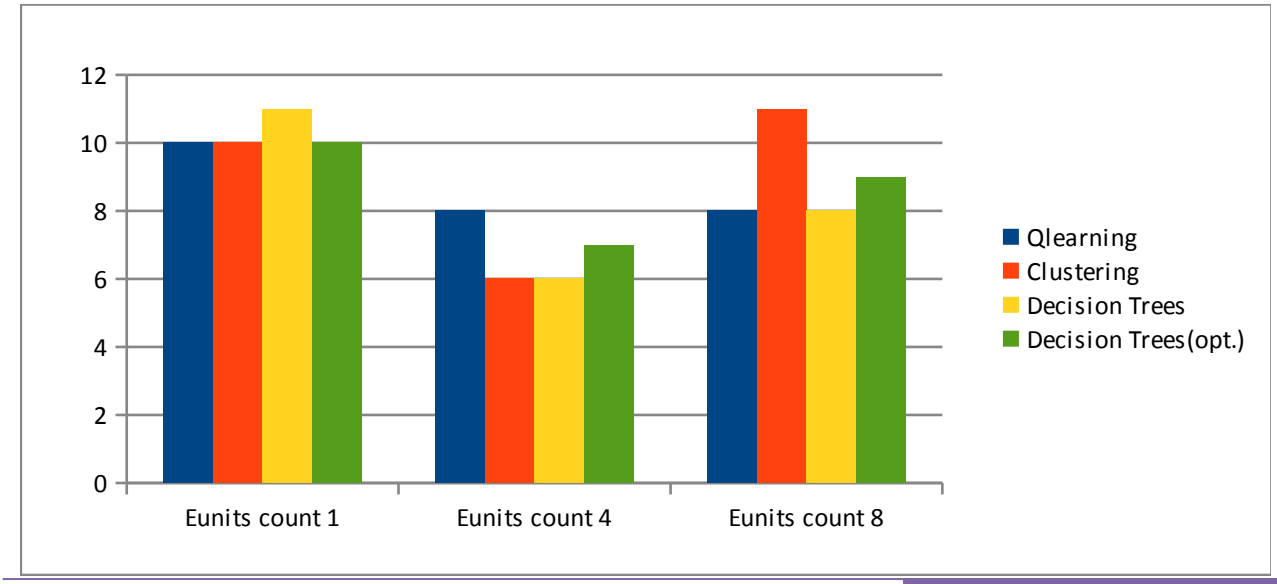

6000
5000
4000
3000
2000
1000
0
Total distance 1
Total distance 4
Total distance 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)

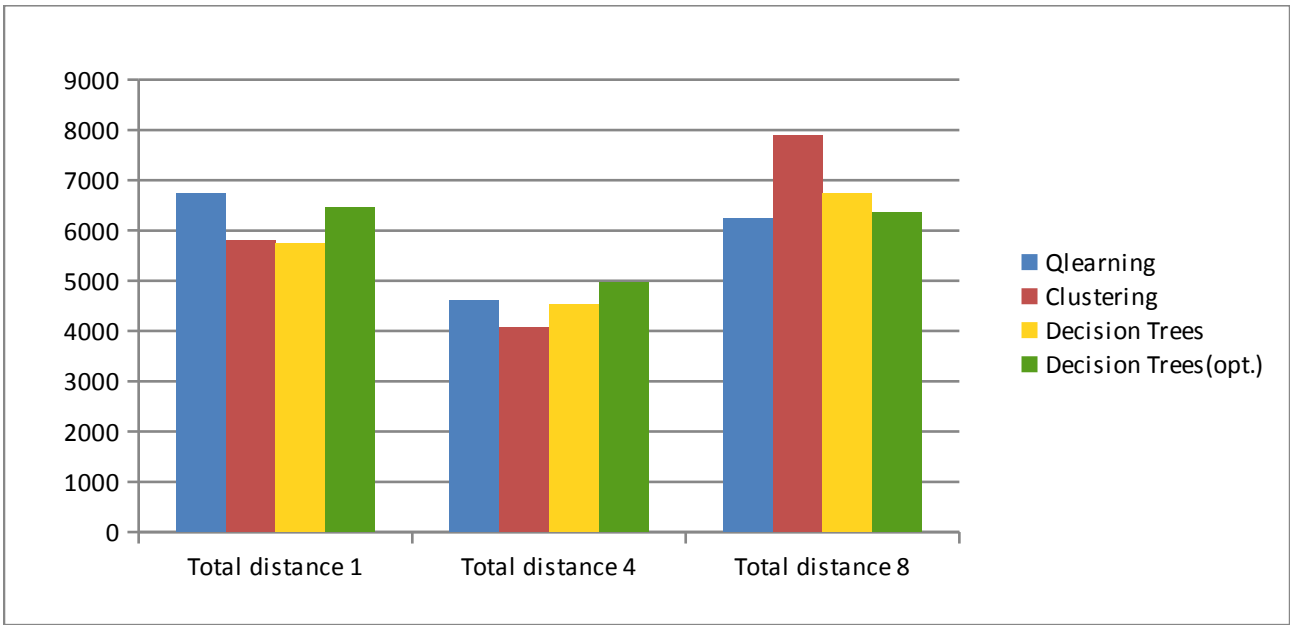

12000
10000
8000
6000
4000
2000
0
Total cost 1
Total cost 4
Total cost 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)

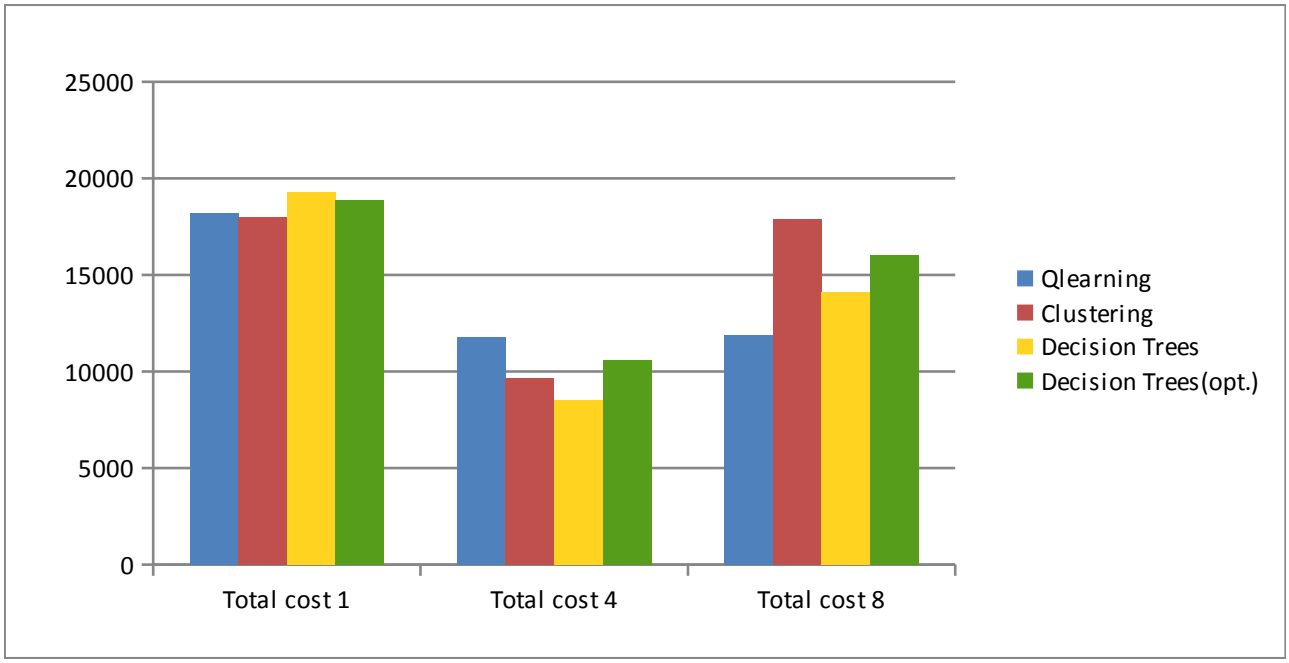


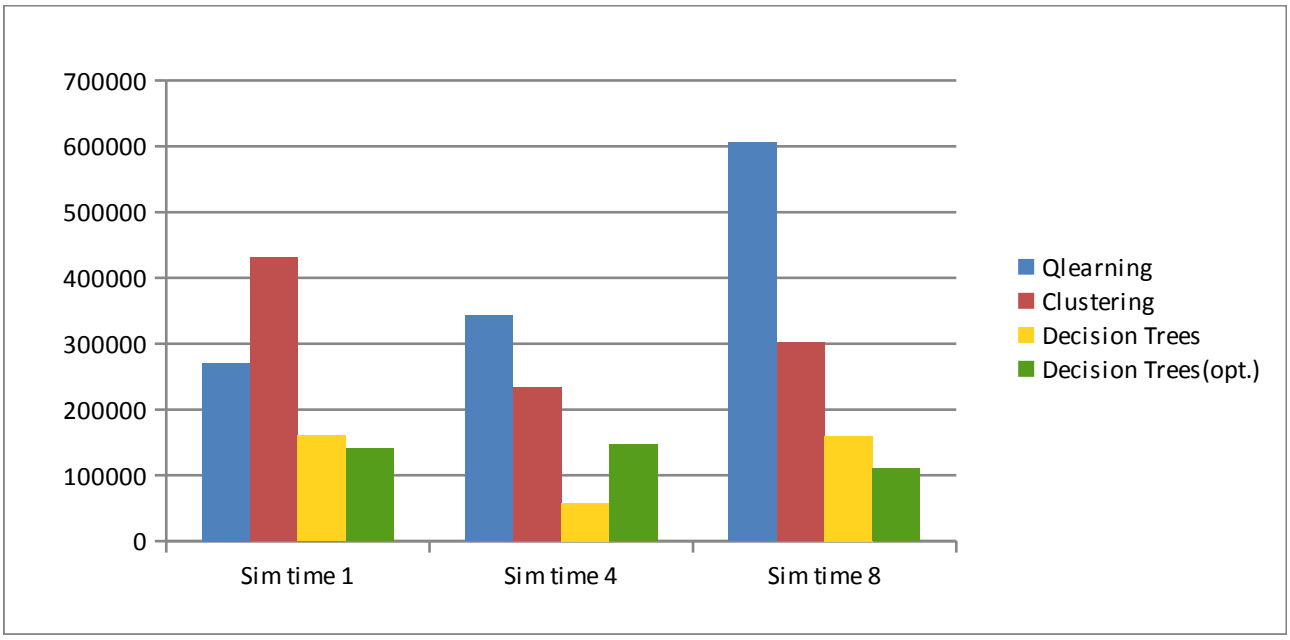
Clustering zachowywał się lepiej dla początkowych pomiarów, dla których symulacja trwała dłużej niż w przypadku QLearningu. Można zauważyć, iż w przypadkach gdy symulacja dla Clusteringu trwała dłużej niż dla QLearningu, wówczas uzyskane wyniki były lepsze. W tym wypadku niestety drzewa decyzyjne przegrały z poprzednimi rozwiązaniami. Użycie optymalnych miar dało nieznacznie lepsze wyniki

4. static, 200, szerokie (lrc221, lrc224, lrc228)



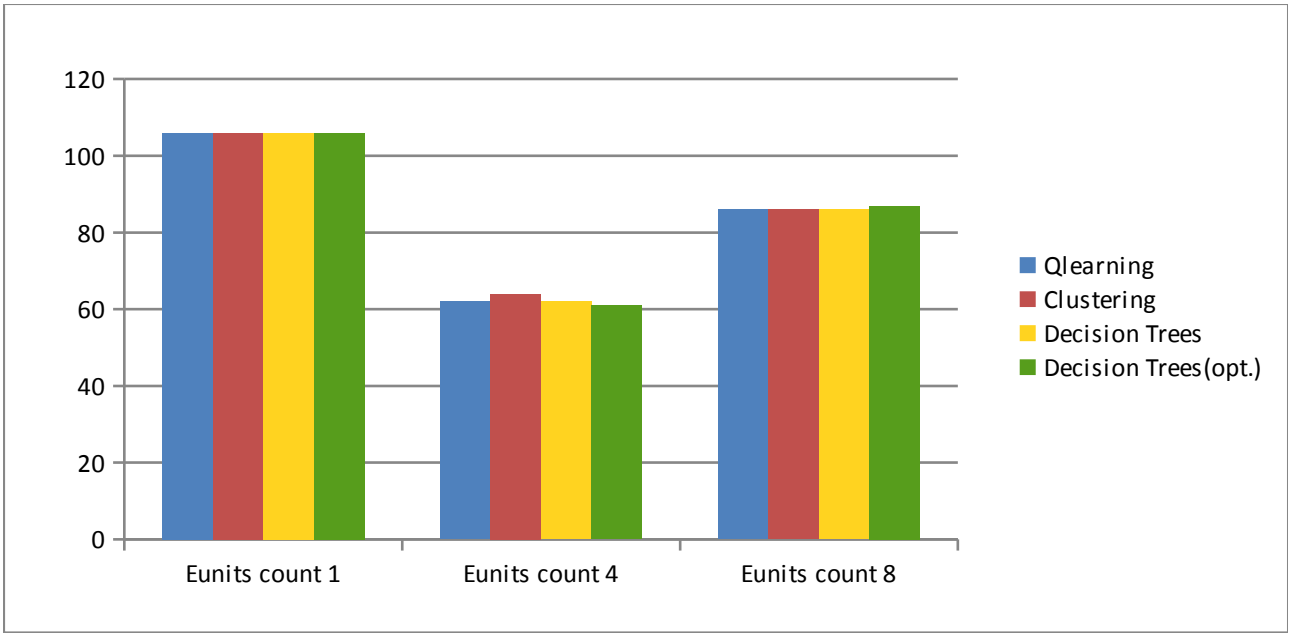

9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Total distance 1
Total distance 4
Total distance 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)

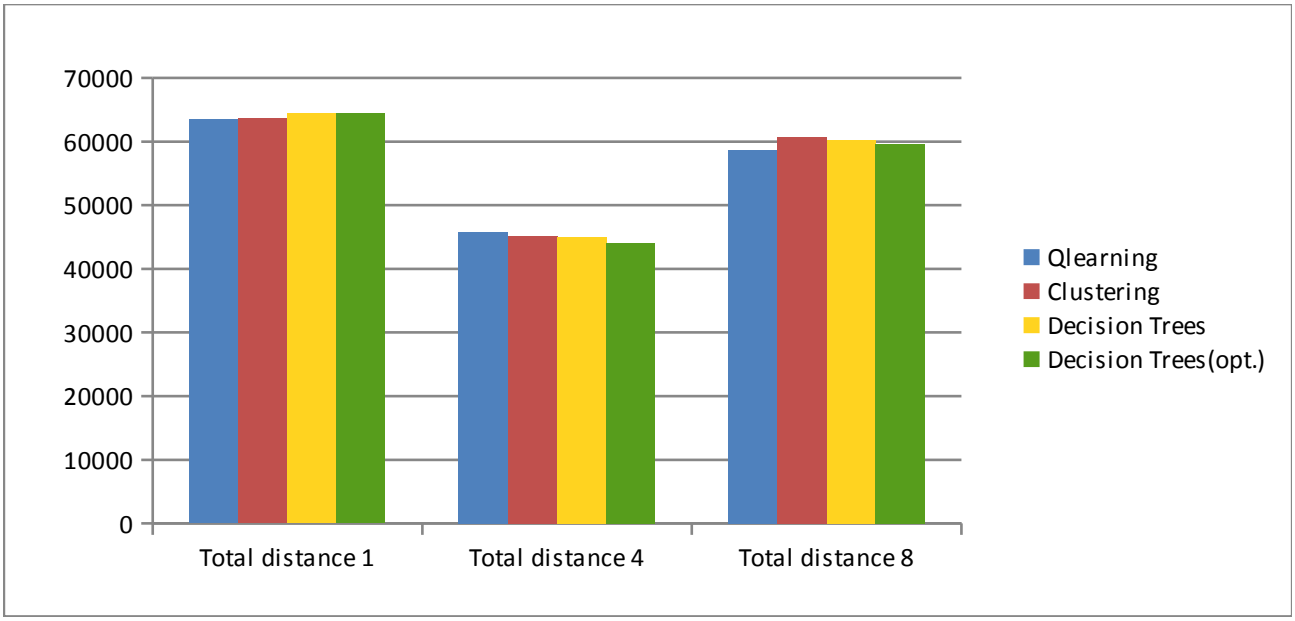

25000
20000
15000
10000
5000
0
Total cost 1
Total cost 4
Total cost 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)

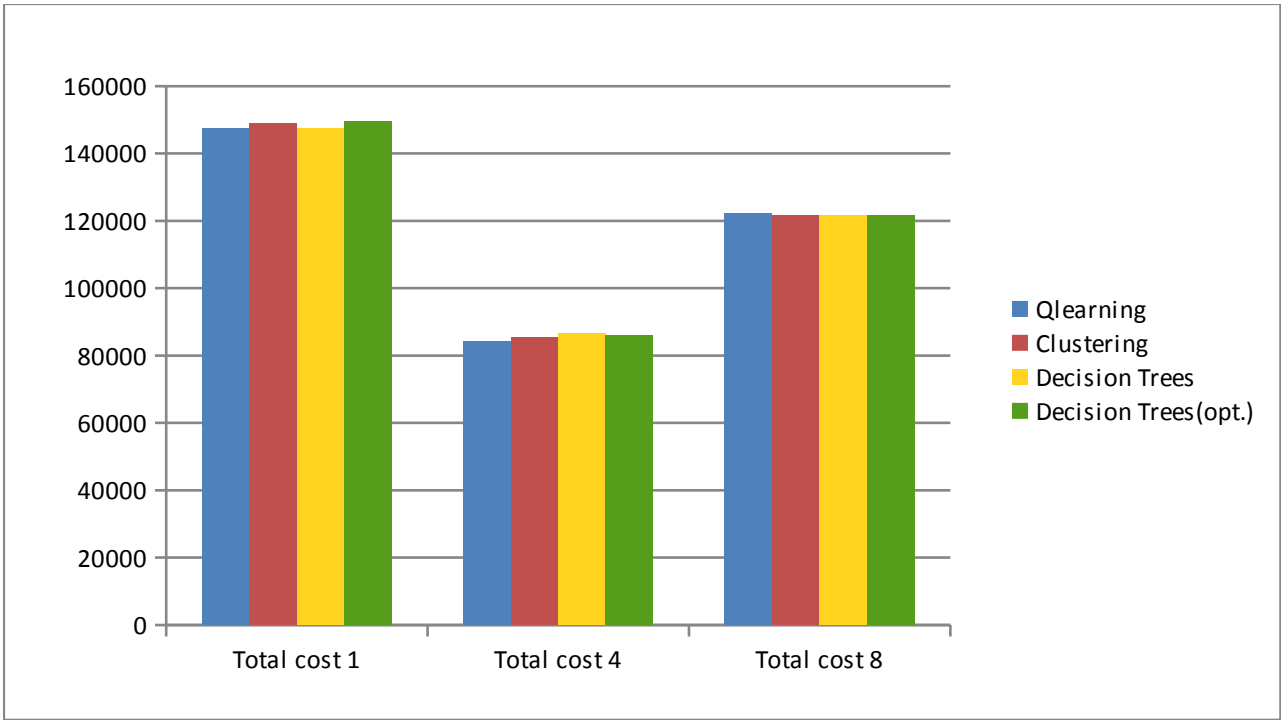


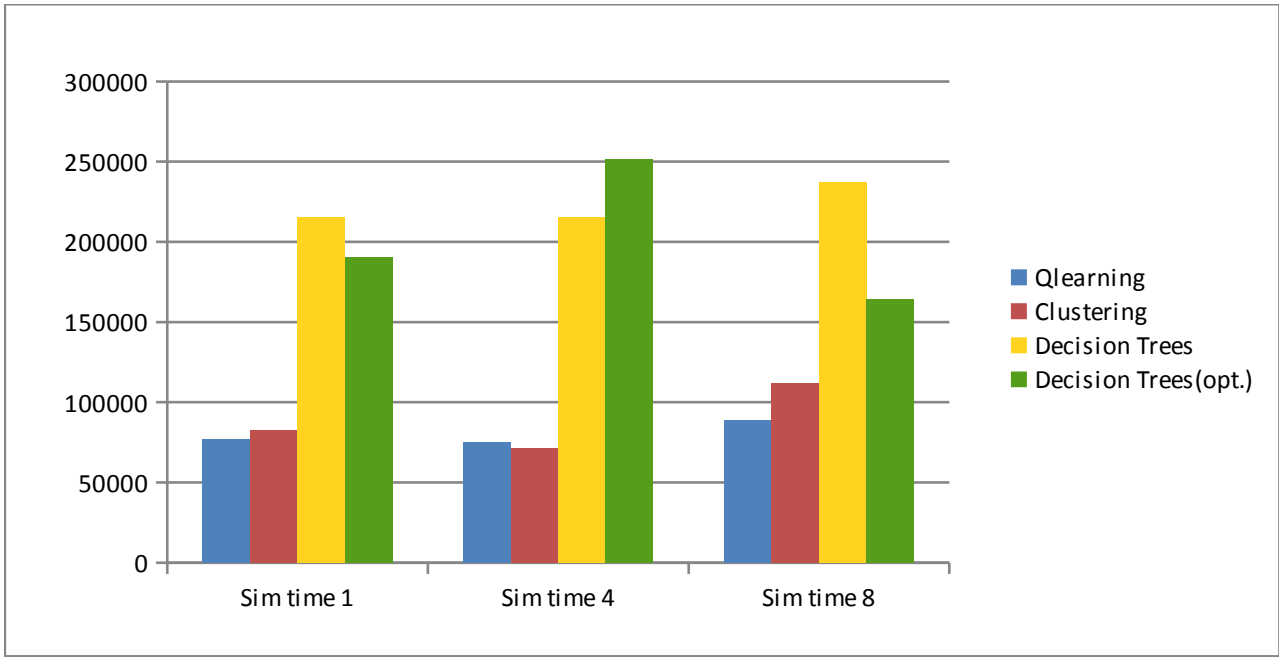
Podobnie jak w p. 3 lepsze wyniki uzyskano w pierwszych dwóch testach. Natomiast w tym wypadku algorytm drzew decyzyjnych sprawował się lepiej niż algorytm Clusteringu. Użycie optymalnych miar nie miało dużego wpływu na wynik, niekiedy nawet go pogorszyło.

5. static, 1000, wąskie (lrc1101, lrc1104, lrc1108)



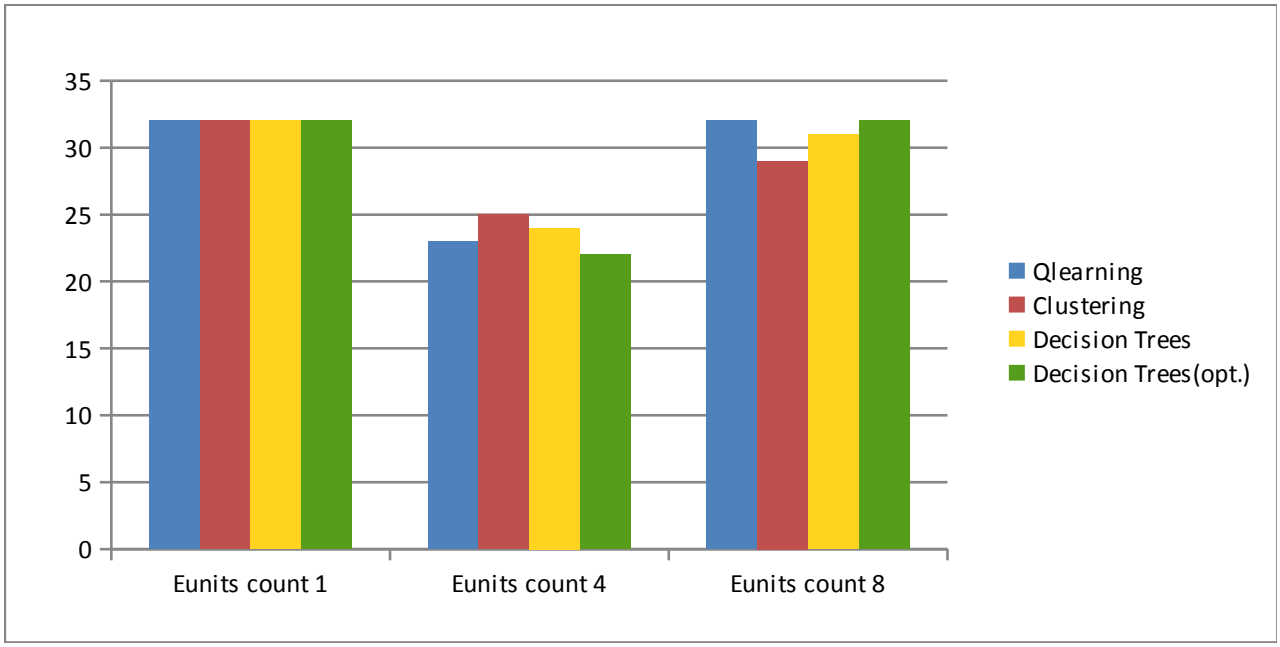

70000
60000
50000
40000
30000
20000
10000
0
Total distance 1
Total distance 4
Total distance 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)

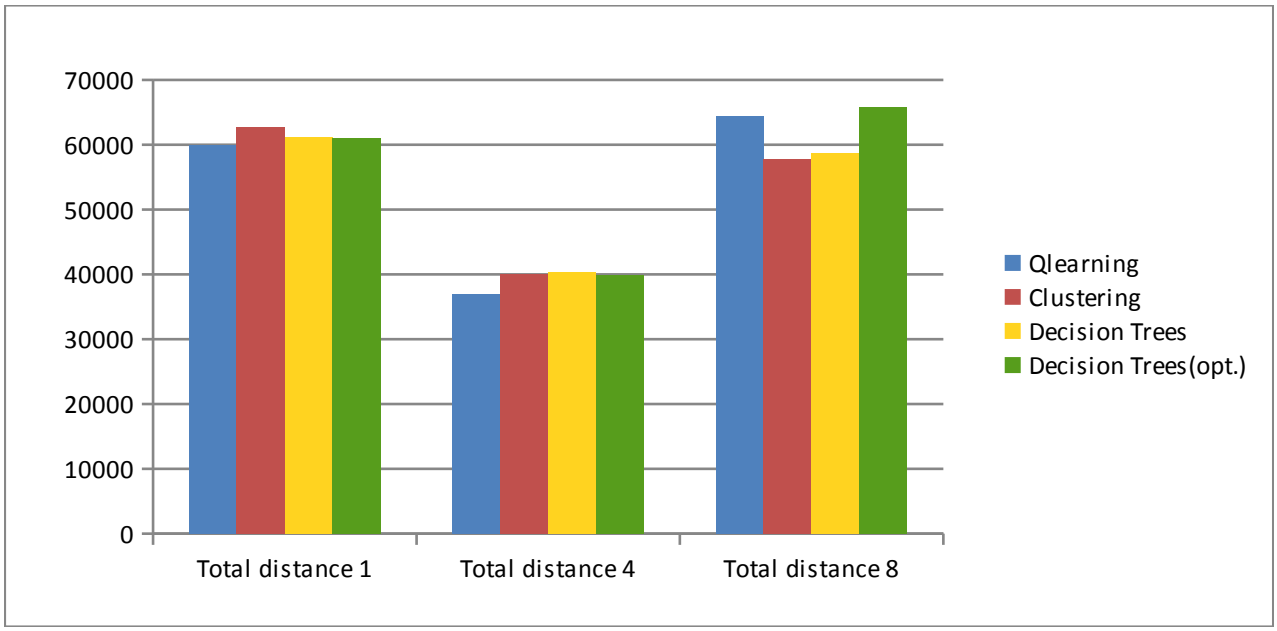

160000
140000
120000
100000
80000
60000
40000
20000
0
Total cost 1
Total cost 4
Total cost 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)

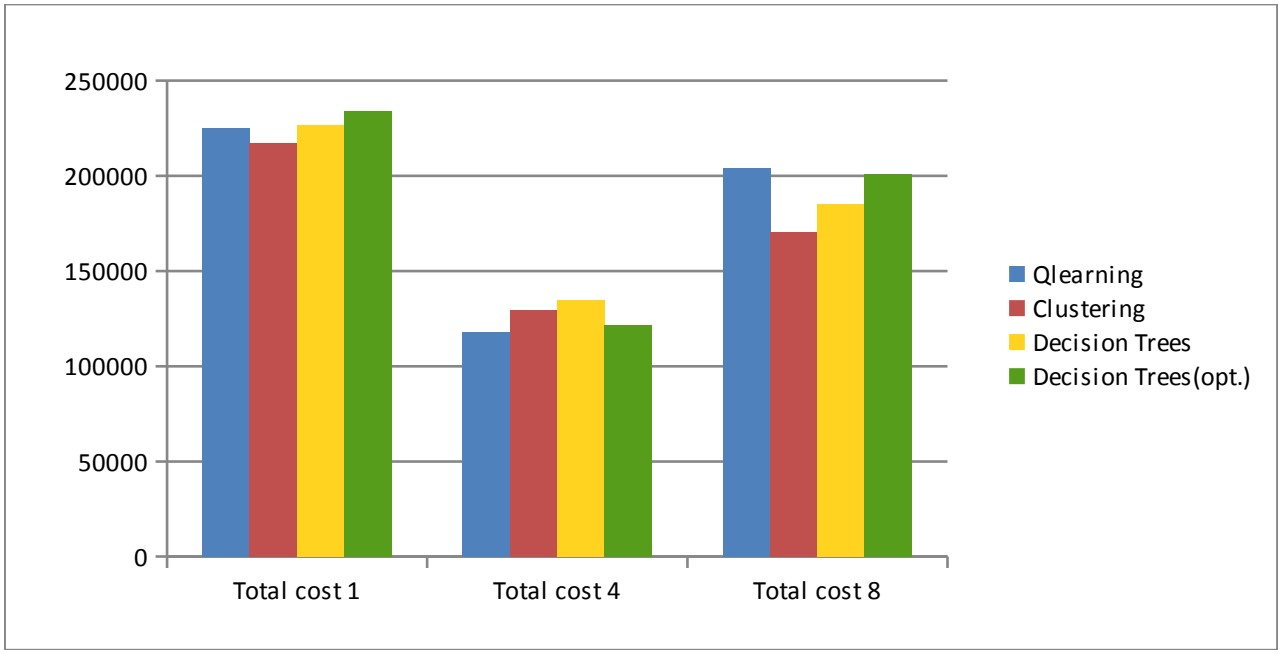


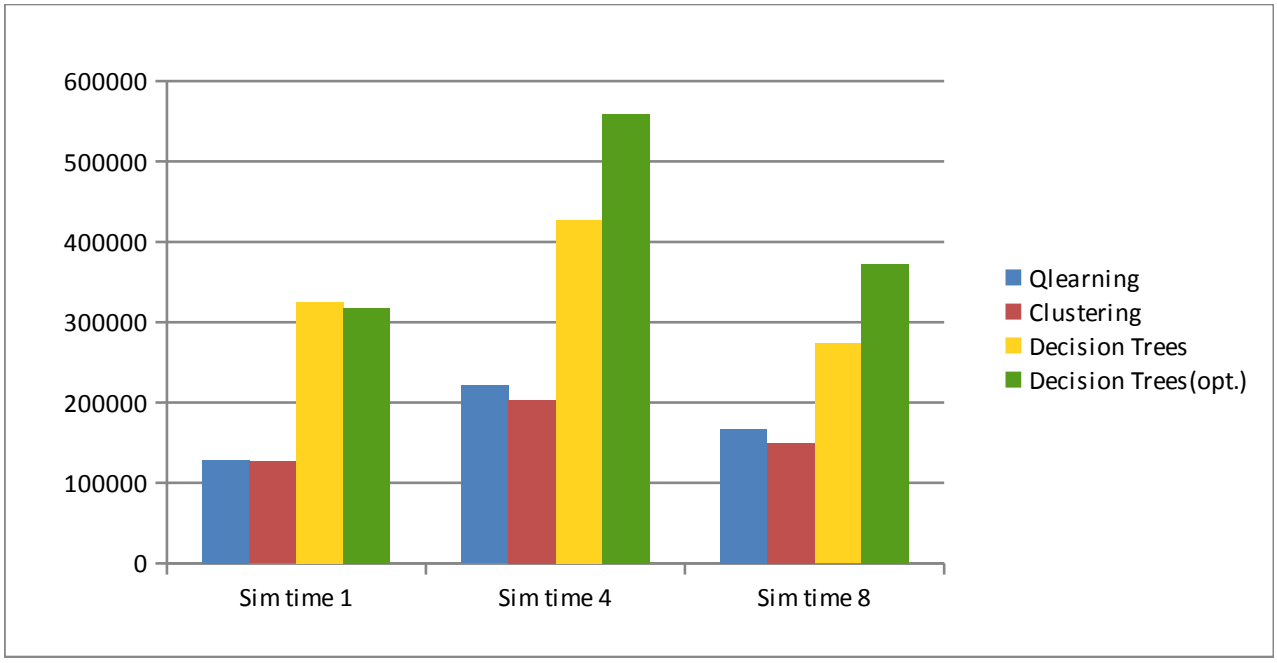
W tym wypadku wyniki były prawie identyczne.

6. static, 1000, szerokie (lrc2101, lrc2104, lrc2108)



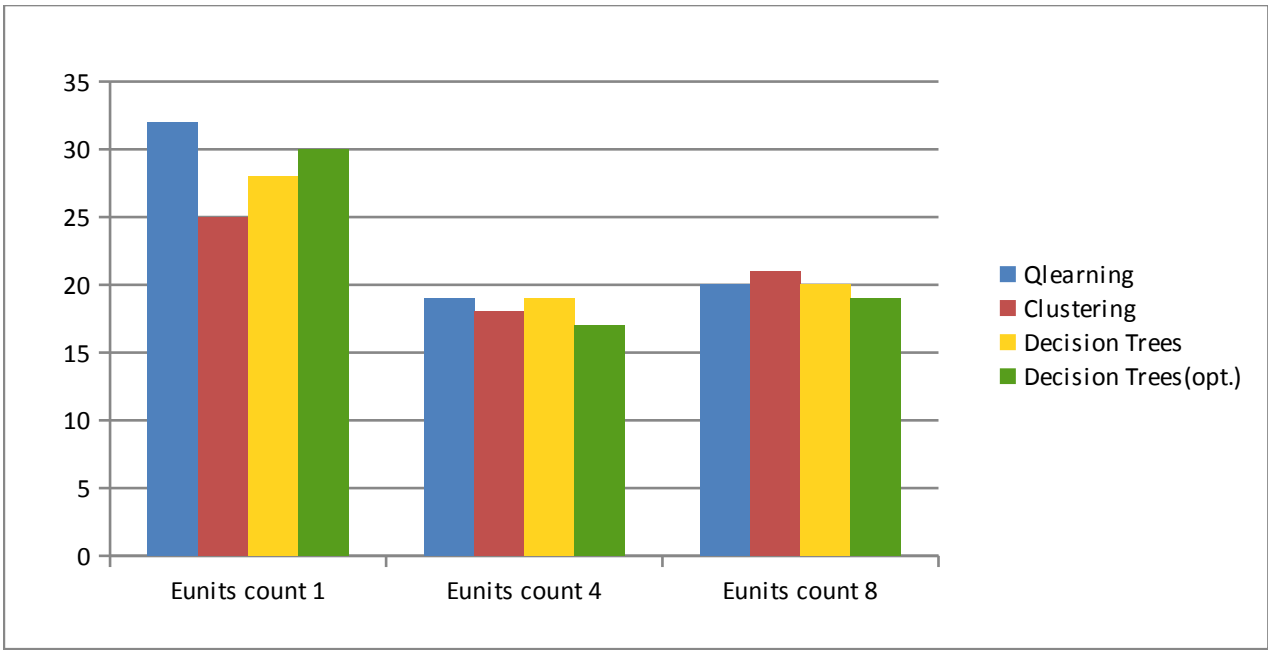

70000
60000
50000
40000
30000
20000
10000
0
Total distance 1
Total distance 4
Total distance 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)

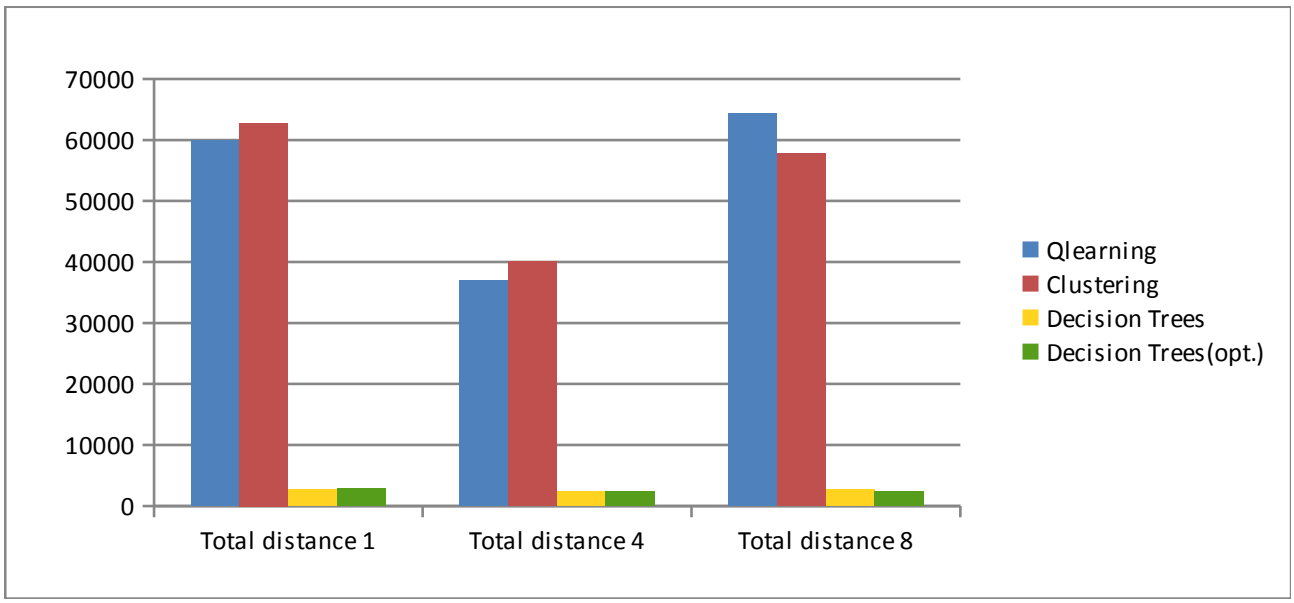

250000
200000
150000
100000
50000
0
Total cost 1
Total cost 4
Total cost 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)

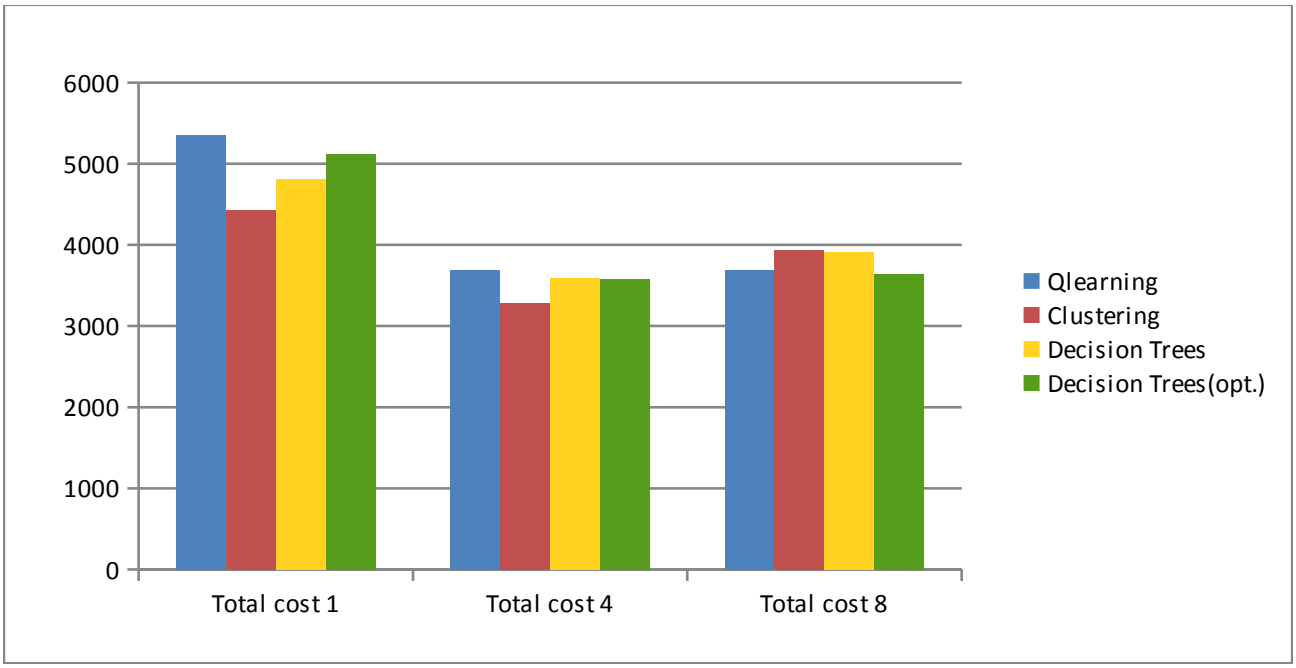


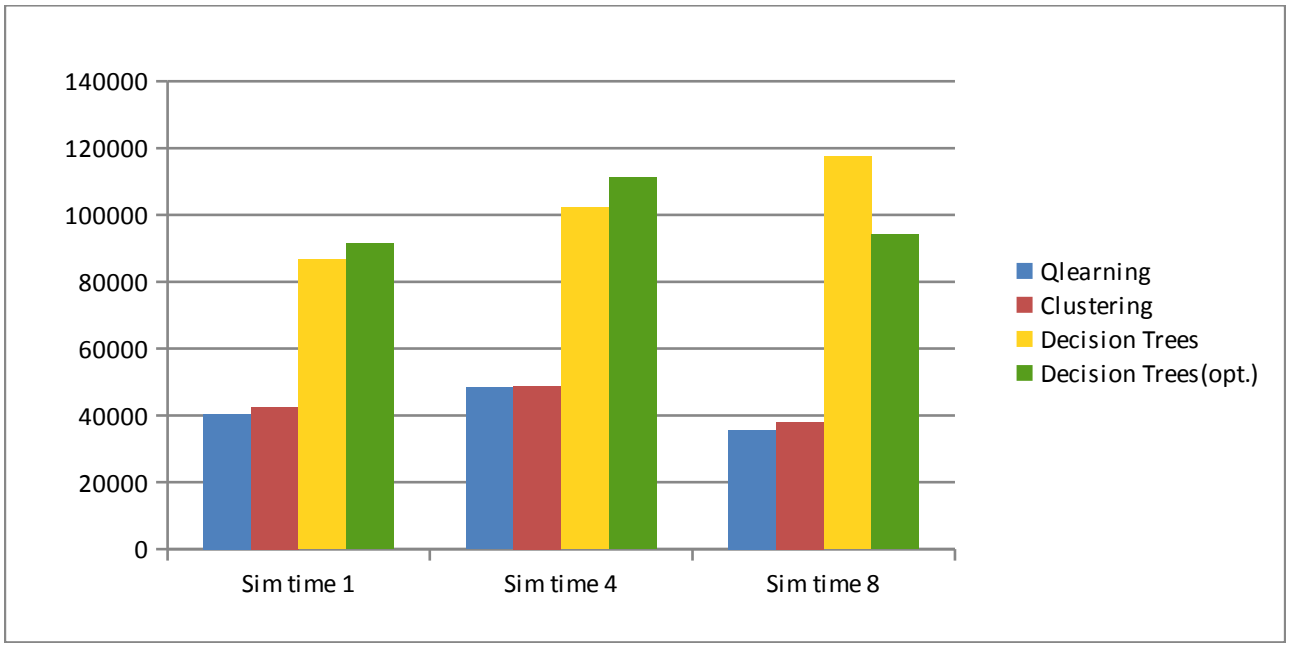
Ponownie jak w p. 1 wraz z kolejnymi testami uzyskujemy lepsze wyniki pomiarów. Co ciekawe wygrywamy również pod względem czasu symulacji (Clustering). Algorytm drzew decyzyjnych dał podobne rezultaty jak Clustering. Użycie optymalnych miar w tym wypadku było nieopłacalne.

7. dynamic, 100, wąskie (lrc101, lrc104, lrc108)



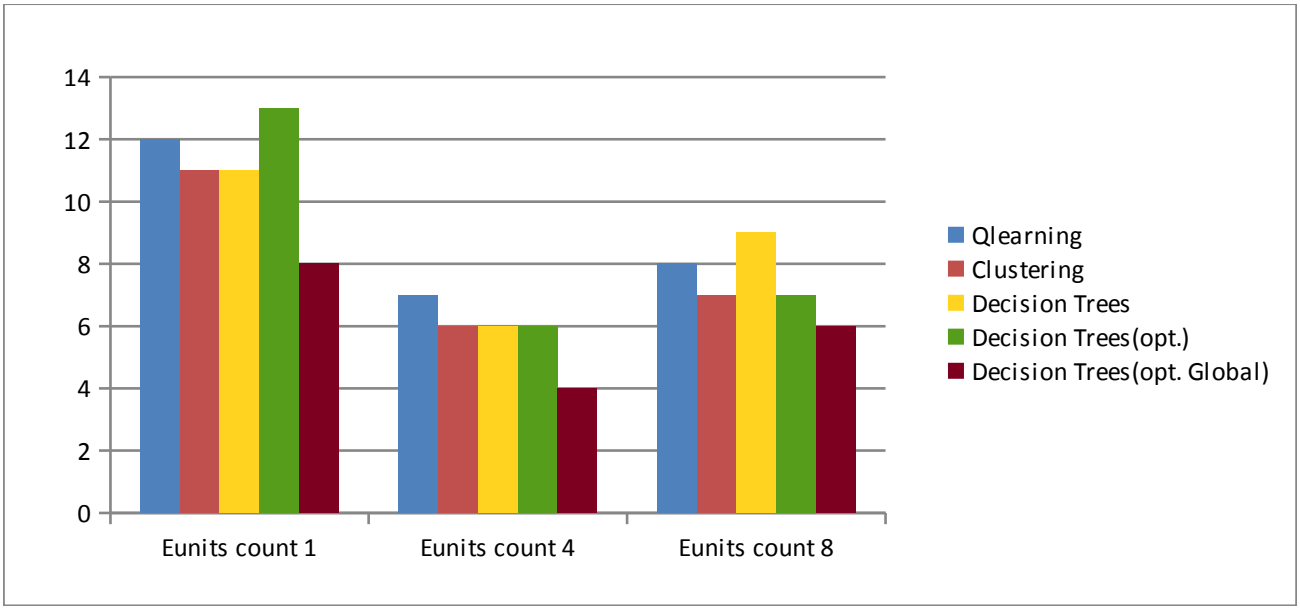

70000
60000
50000
40000
30000
20000
10000
0
Total distance 1
Total distance 4
Total distance 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)

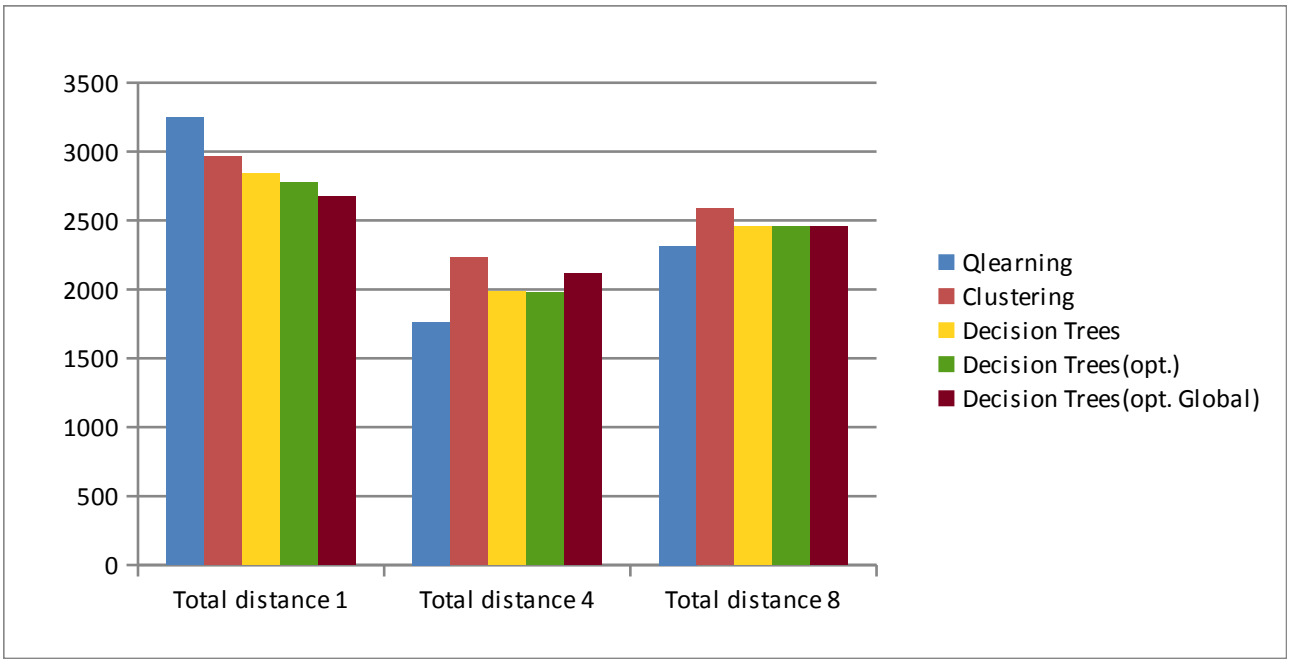

6000
5000
4000
3000
2000
1000
0
Total cost 1
Total cost 4
Total cost 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)

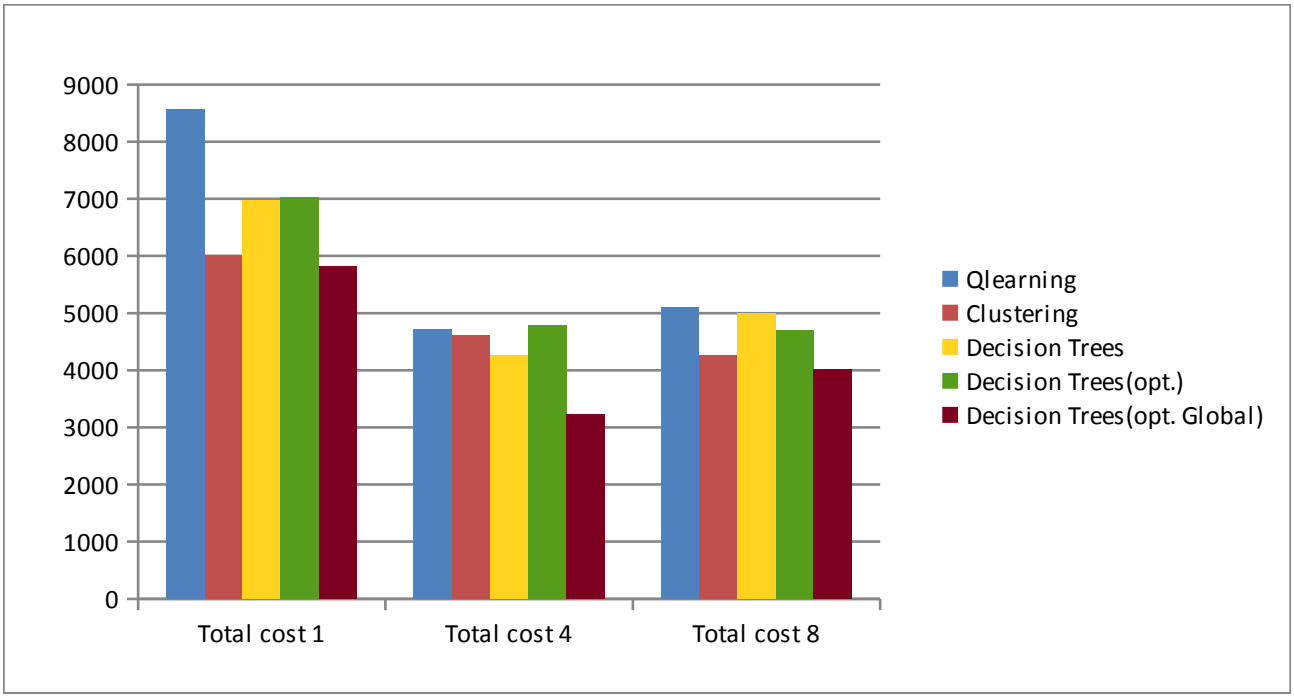


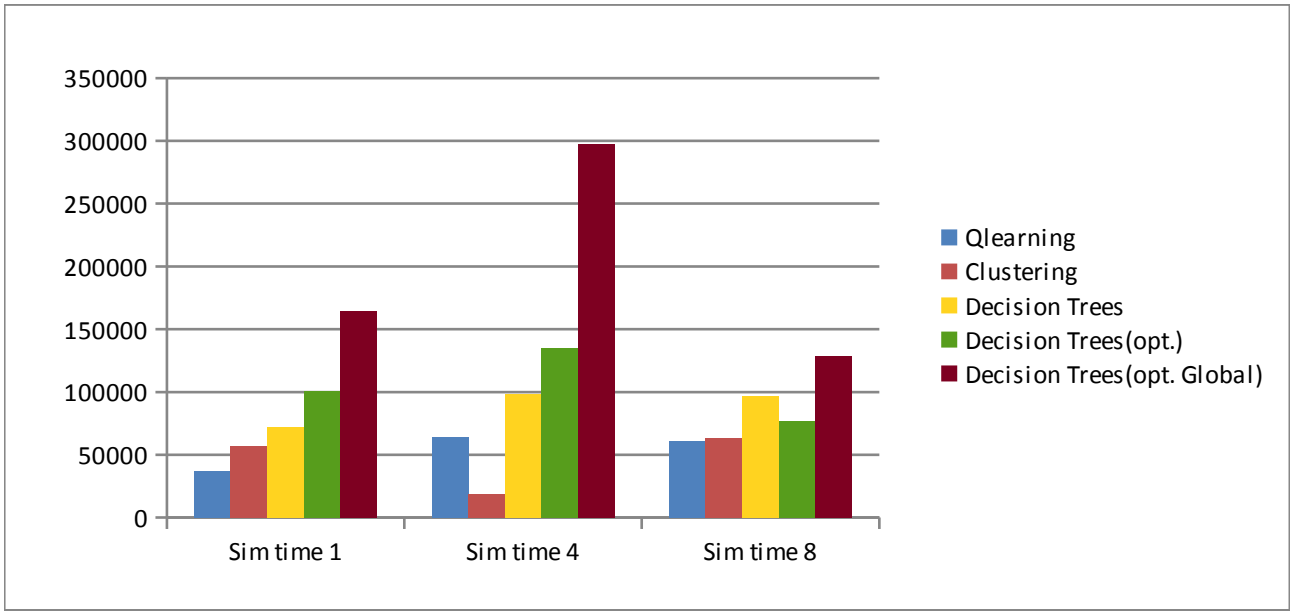
Algorytm Clusteringu nie zawsze ale minimalnie wygrywa z QLearningiem, w obu przypdkach czas symulacji jest bardzo podobny. Drzewa decyzyjne podobnie jak Clustering minimalnie wygrywają z QLearningiem. Użycie optymalnych miar dało nieznacznie lepsze wyniki.

8. dynamic, 100, szerokie (lrc201, lrc204, lrc208)



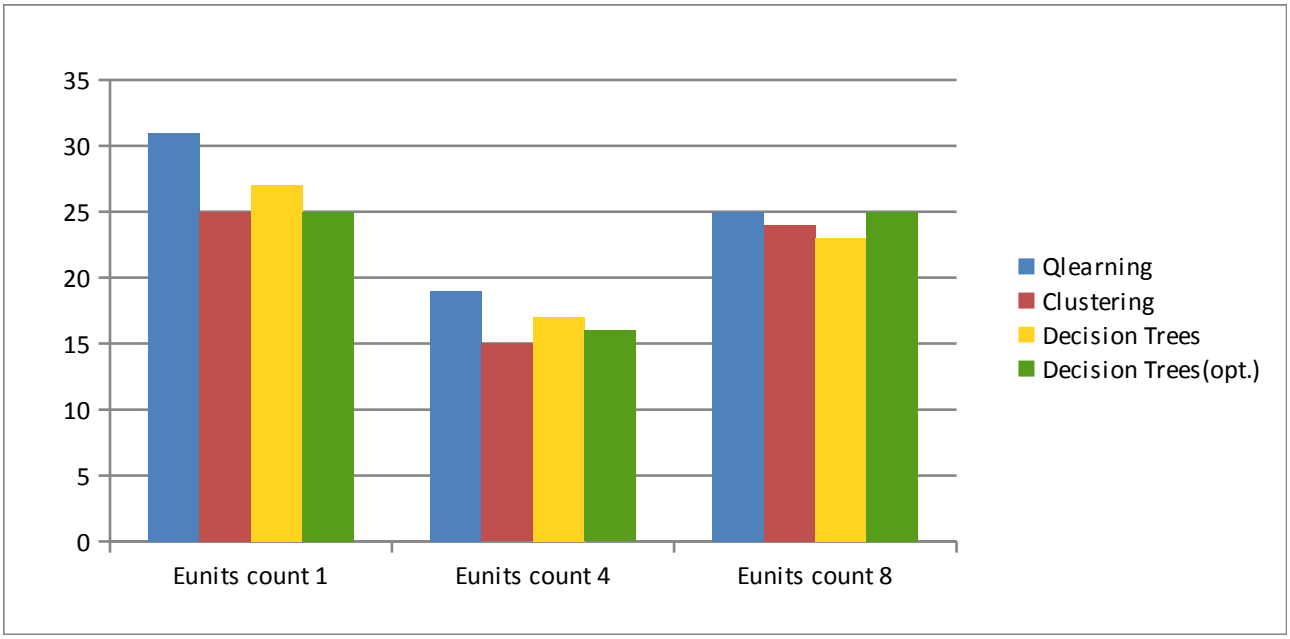

3500
3000
2500
2000
1500
1000
500
0
Total distance 1
Total distance 4
Total distance 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)
Decision Trees(opt. Global)

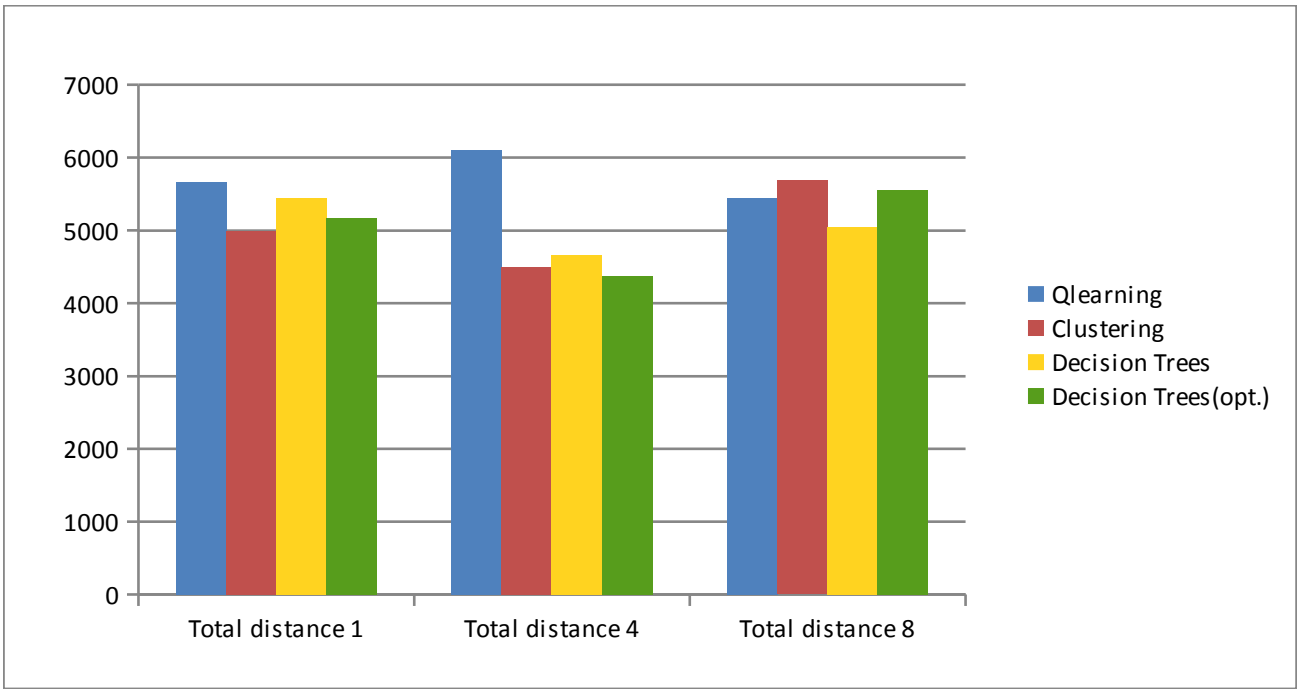

9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Total cost 1
Total cost 4
Total cost 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)
Decision Trees(opt. Global)

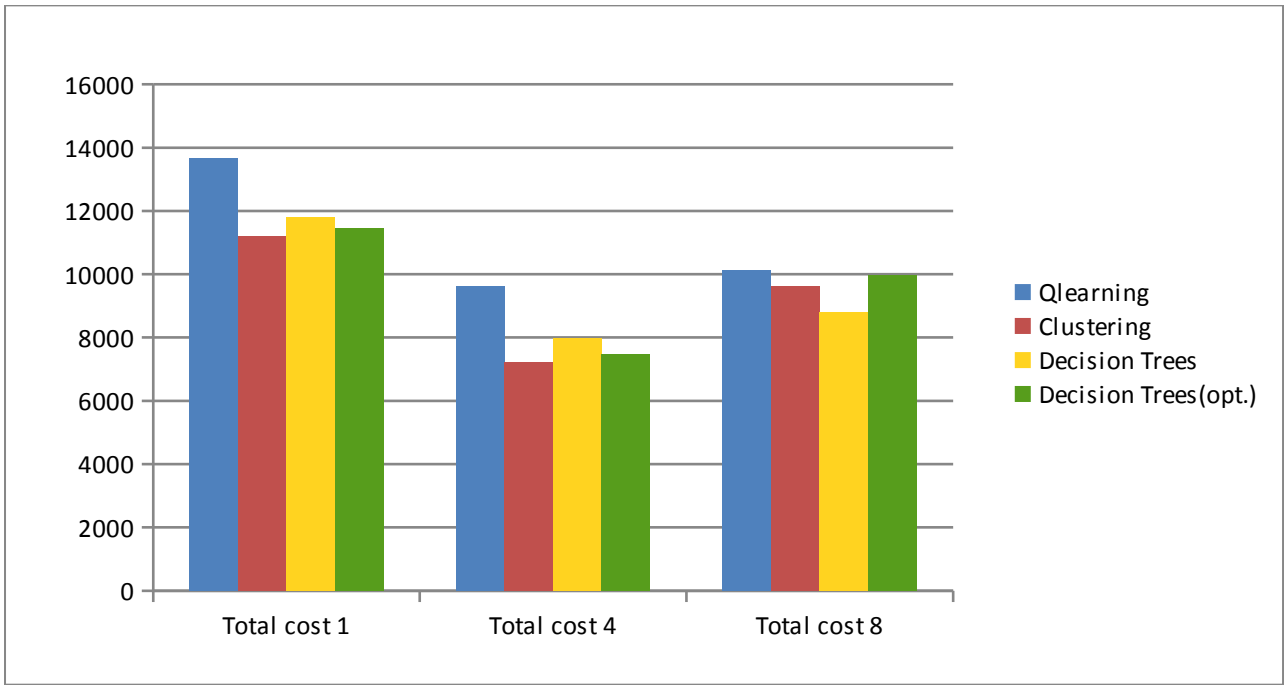


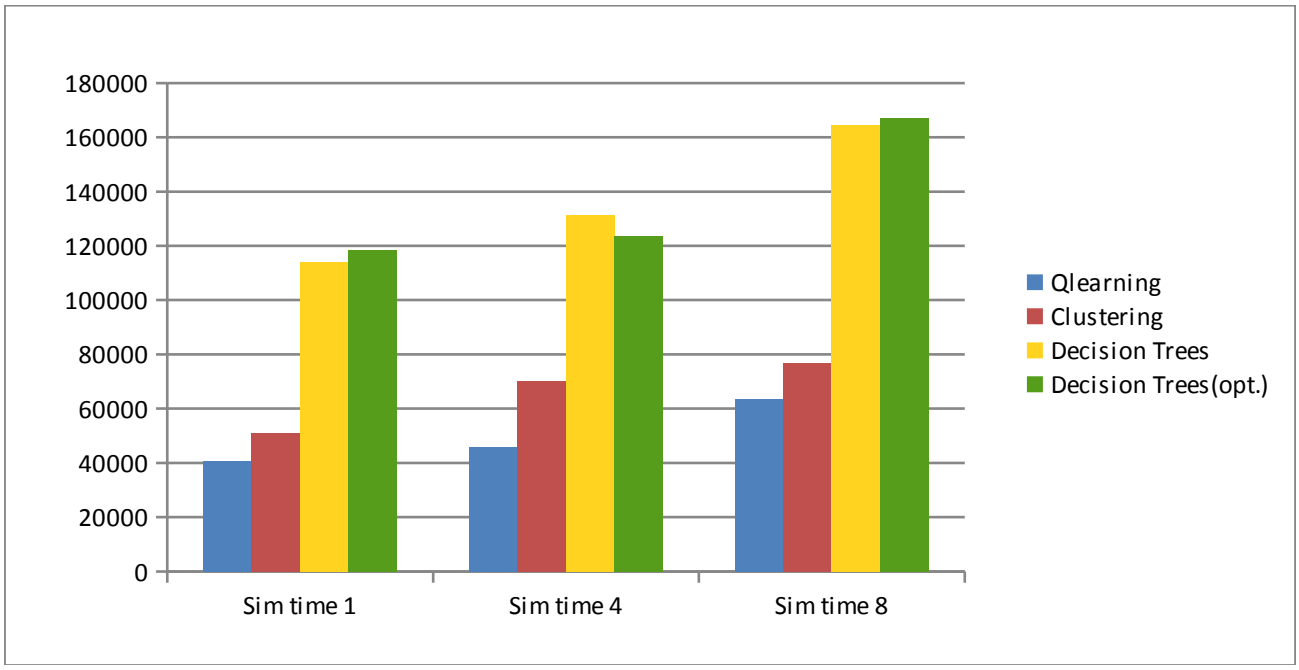
Przypadek w którym całkowity przejechany dystans jest wyższy w przypadku Clusteringu, jednak co ważne w każdym z pomiarów uzyskujemy niższy koszt. Interesujący jest wyniki dla lrc204 gdzie uzyskujemy znacznie niższy czas symulacji. Drzewa decyzyjne jak w większości przypadków zachowują się lepiej niż Clustering. Użycie optymalnych miar nie miało wpływu na wynik. Wyłączenie zmian konfiguracji dla holonów było dobrym pomysłem i dało bardzo dobre rezultaty (najlepsze wyniki).

9. dynamic, 200, wąskie (lrc121, lrc124, lrc128)



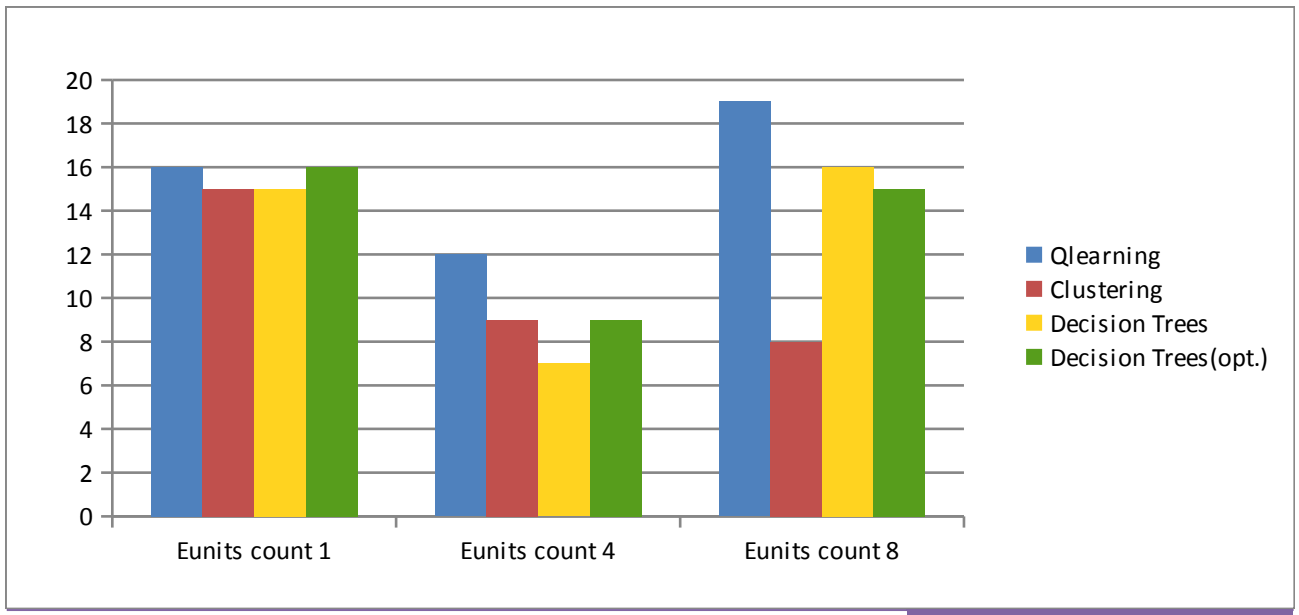

7000
6000
5000
4000
3000
2000
1000
0
Total distance 1
Total distance 4
Total distance 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)

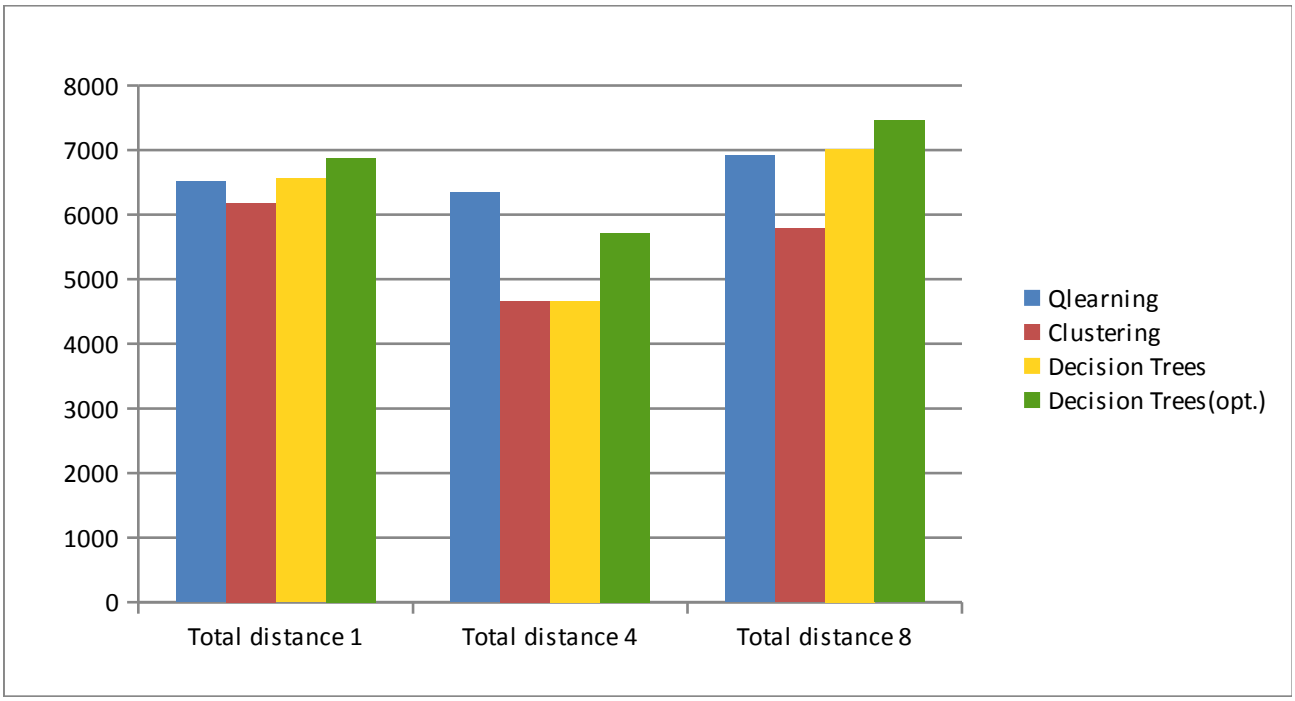

16000
14000
12000
10000
8000
6000
4000
2000
0
Total cost 1
Total cost 4
Total cost 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)

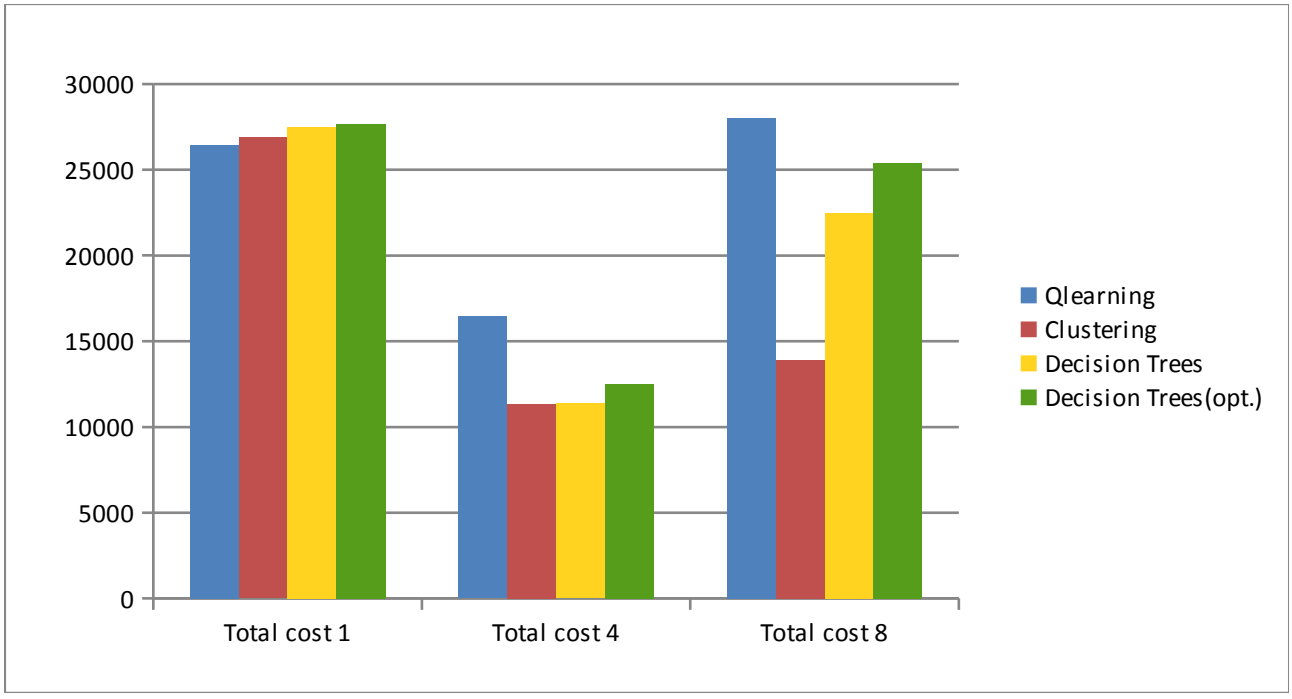


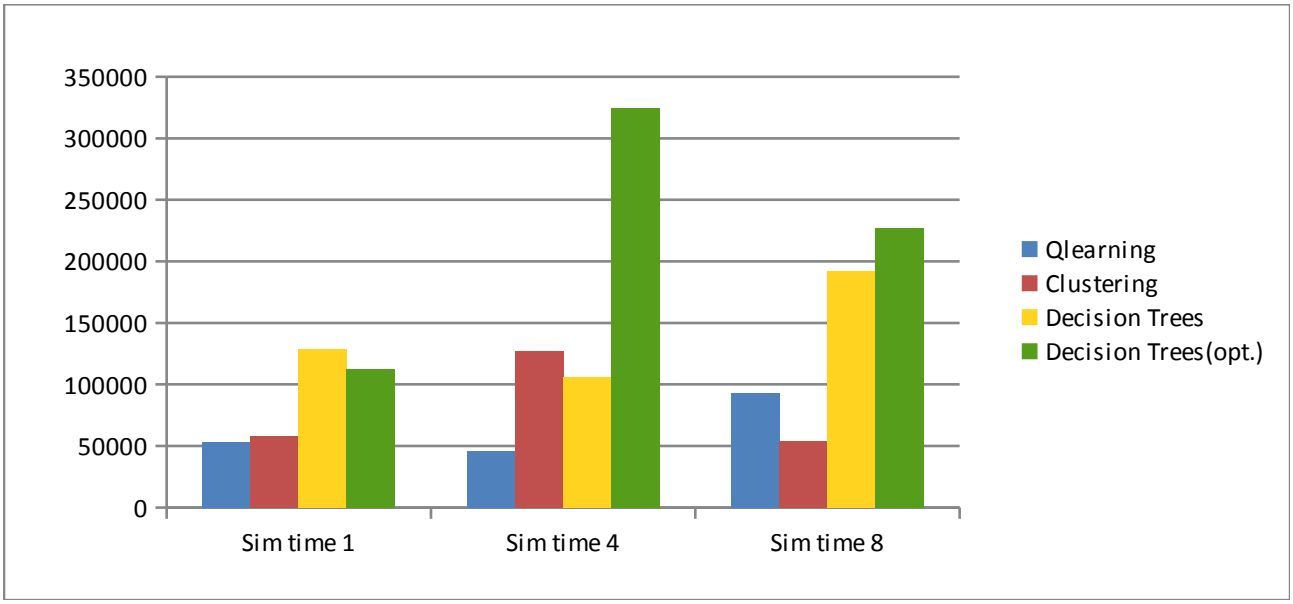
Podobnie jak w p.8, przy czym czas symulacji dla Clusteringu jest nieznacznie wyższy. Drzewa decyzyjne dają lepsze wyniki niż Qlearning, natomiast gorze niż Qlearning. Użycie optymalnych miar nie miało wpływu na wynik.

10. dynamic, 200, szerokie (lrc221, lrc224, lrc228)




8000
7000
6000
5000
4000
3000
2000
1000
0
Total distance 1
Total distance 4
Total distance 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)


30000
25000
20000
15000
10000
5000
0
Total cost 1
Total cost 4
Total cost 8
Qlearning
Clustering
Decision Trees
Decision Trees(opt.)



Najlepszy z naszych wyników, szczególnie w przypadku lrc228, wygrywamy w każdym aspekcie łącznie z czasem symulacji. Warto zaznaczyć, że wyniki dla Clusteringu są o wiele lepsze, w przypadku lrc228 uzyskujemy dwa razy niższy koszt. Drzewa decyzyjne podobnie jak wyżej, lepsze od QLearningu, gorsze od Clusteringu. W tym szczególnym wypadku użycie optymalnych miar pogorszyło nieznacznie wynik.

9. Testy miar

9.1 Realizacja

Spośród wszystkich dostępnych miar, następujące zostały wybrane do przeprowadzenia testów:

1. AverageLoadFromAllCommissions
2. AverageMaxTimeWinSizeForAllCommissions
3. AverageMinTimeWinSizeForAllCommissions
4. AverageMinDistBetweenAllCommissions
5. PercentageOfDelayComs
6. WaitTime
7. MaxWaitTime

Miary odrzucone:

1. AverageDistanceFromCurLocationToBaseForAllCommissions, AverageDistanceFromCurLocationToBaseForUndeliveredCommissions - trudno na podstawie tych miar wysnuć jakieś wnioski dotyczące jakości symulacji, miary są ściśle uzależnione od tego jakie zlecenia zostały przydzielone, a na to nie ma wpływu
2. AverageDistPerCommissionBeforeChange - używana wersja After
3. AverageLatencyPerCommission - prawdopodobnie błąd w algorytmie, dodawane są do siebie odległość i czas, błąd znajduje się w metodzie measure.AverageLatencyPerCommission.calculateAverageLatency(Schedule)
4. AverageLoadFromUndeliveredCommissions - wybrano wersję for AllCommisions
5. AverageMaxTimeWinSizeForUndeliveredCommissions,AverageMinTimeWinSizeForUndeliveredCommissions - wybrano wersję max i po all commisions
6. AverageMinDistBetweenUndeliveredCommissions - wybrano wersję for all
7. AverageNumberOfComsWithinTimeWinOfAllCommissions,AverageNumberOfComsWithinTimeWinOfUndeliveredCommissions - wystarczające jest użycie miary max time windows size
8. AverageTimeWindowsSizeForAllCommissions,AverageTimeWindowsSizeForUndeliveredCommissions - liczone max
9. DistFromCenterOfGravityOfAllCommissionsToHolon,DistFromCenterOfGravityOfUndeliveredCommissionsToHolon - jak w 1.
10. GivenCommissionsNumber – algorytm nie ma na to wpływu, więc nie niesie dla symulacji żadnej potrzebnej informacji
11. MaxLatency - jak w 3.
12. NumberOfCommissions, NumberOfCommissionsOthersCanAddToUsAfterChanges, NumberOfCommissionsOthersCanAddToUsBeforeChanges, NumberOfCommissionsWeCanAddToOthersAfterChanges, NumberOfCommissionsWeCanAddToOthersBeforeChanges - nie niesie żadnej sensownej informacji
13. ReceivedCommissionsNumber - j.w.
14. miary liczące odchylenie zostały odrzucone ponieważ używamy max/min
15. SummaryLatency - jak w 3.
16. AverageDistPerCommissionAfterChange <- nie występuje w runtime

Metodyka testów miar:

1. Dodajemy wszystkie miary
2. Przeprowadzamy symulację (jeden rozmiar problemu, jeden przebieg nauczenia, jeden przebieg jałowy dla uzupełnienia tablicy, raz dla problemu statycznego, raz dla dynamicznego, w efekcie 6 symulacji w jednym kroku)
3. Odejmujemy jedną miarę
4. Powtarzamy symulację z 2.
5. Jeśli wynik jest lepszy - odrzucamy miarę, jeśli gorszy - zostawiamy
6. Powtarzaj do momentu wyczerpania wszystkich możliwości (odjęcie jakiejkolwiek miary nie poprawia wyniku)

Testy przeprowadzono dla następujących benchmarków :

- lrc208 = static, 100, szerokie
- lrc128 = dynamic, 200, wąskie (dynamicB)

Do konfiguracji globalnej użyto wartości średniej dla podanych measurmentów ponieważ wynikiem musi być jedna liczba.

Np. dla measurmentów w konfiguracji holonicznej :

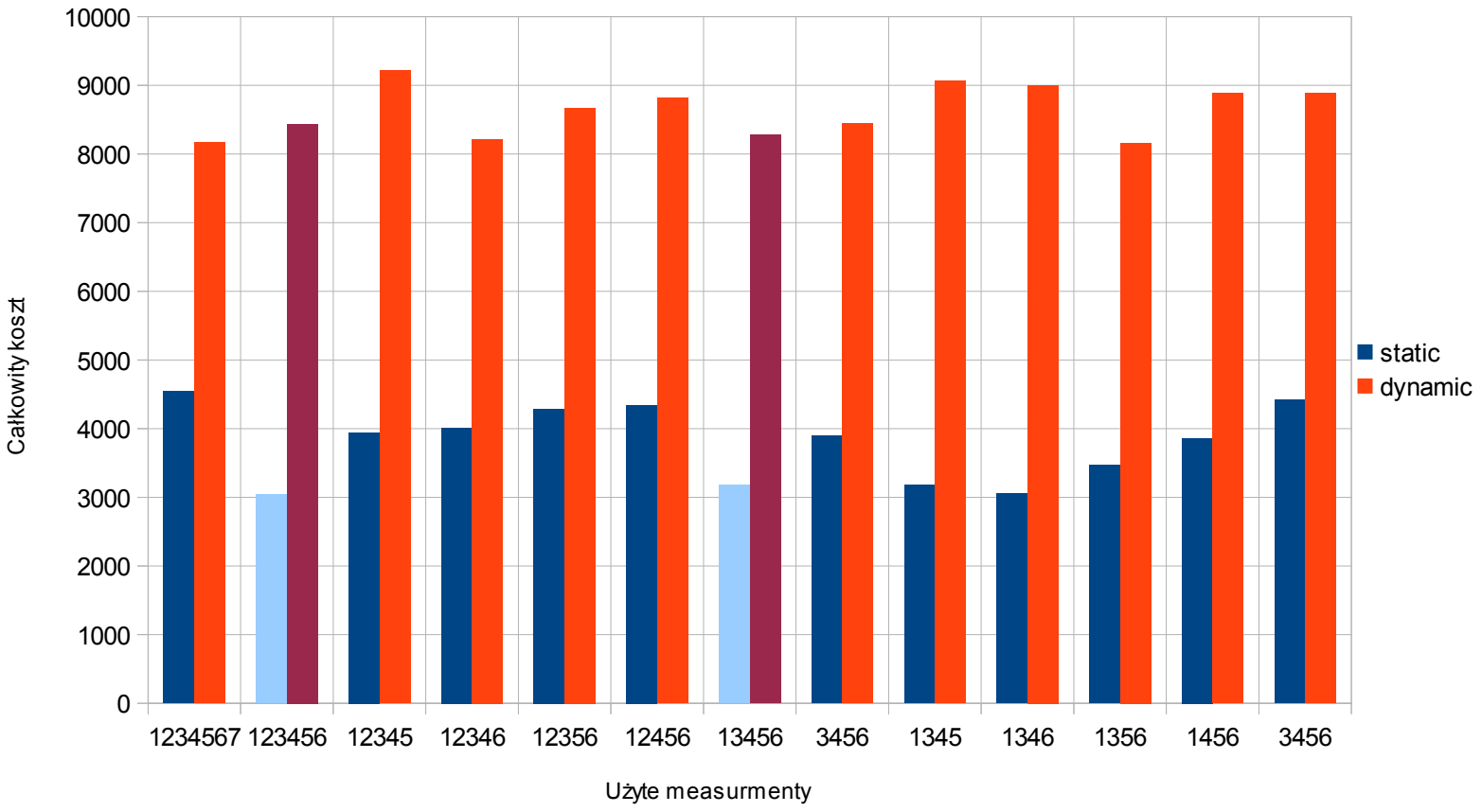
```
<holonMeasures>
    <measure name="M1" value="AverageMinTimeWinSizeForAllCommissions"/>
    <measure name="M2" value="AverageMinDistBetweenAllCommissions"/>
    <measure name="M3" value="PercentageOfDelayComs"/>
    <measure name="M4" value="WaitTime"/>
</holonMeasures>
```

Globalne measurmenty miały następujące wartości :

```
<globalMeasures>
    <measure name="M1" value="avg(AverageMinTimeWinSizeForAllCommissions)"/>
    <measure name="M2" value="avg(AverageMinDistBetweenAllCommissions)"/>
    <measure name="M3" value="avg(PercentageOfDelayComs)"/>
    <measure name="M4" value="avg(WaitTime)"/>
</globalMeasures>
```

Opis plików z wynikami :
Pliki z wynikami dla poniższych testów znajdują się w folderze „testyMiar”, w którego wnętrzu znajdują się foldery, których nazwa oznacza „numerTestu_numeryUżytychMiar”. W każdym z powyższych folderów znajdują się dwa pliki z wynikami, jeden dla problemu statycznego, drugi dla dynamicznego.

9.2 Wyniki



Opis wykresu :
Powyższy wykres pokazuje jak kształtowała się wartość całkowitego kosztu podczas przeprowadzania testów. Wartość na osi X oznacza sekwencję miar użytych w danym teście.

Spośród wszystkich dostępnych miar, następujące zostały wybrane do przeprowadzenia testów:

1. AverageLoadFromAllCommissions
2. AverageMaxTimeWinSizeForAllCommissions
3. AverageMinTimeWinSizeForAllCommissions
4. AverageMinDistBetweenAllCommissions
5. PercentageOfDelayComs
6. WaitTime
7. MaxWaitTime

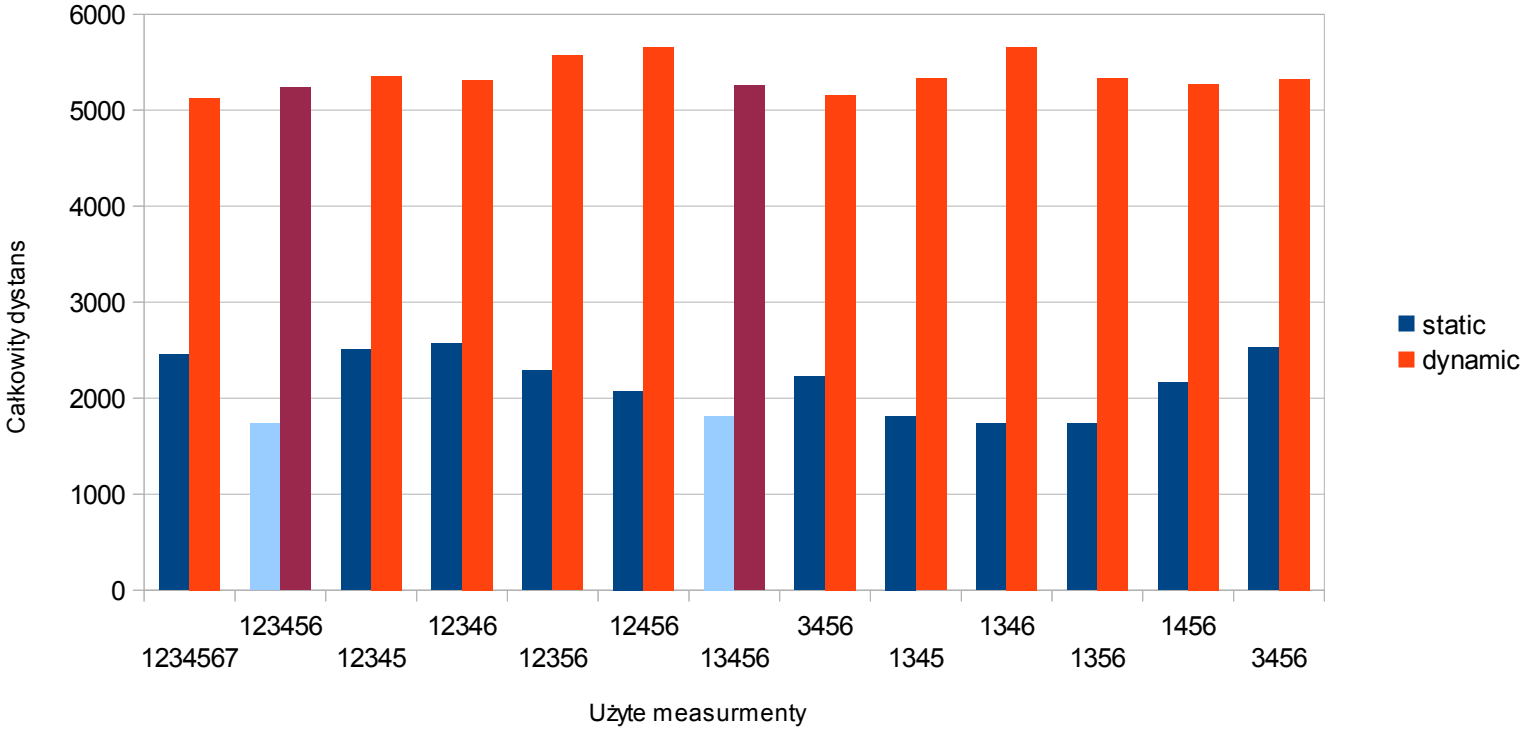
Np. 1346 oznacza, że do testów zostały użyte miary o numerach 1,3,4 oraz 6.

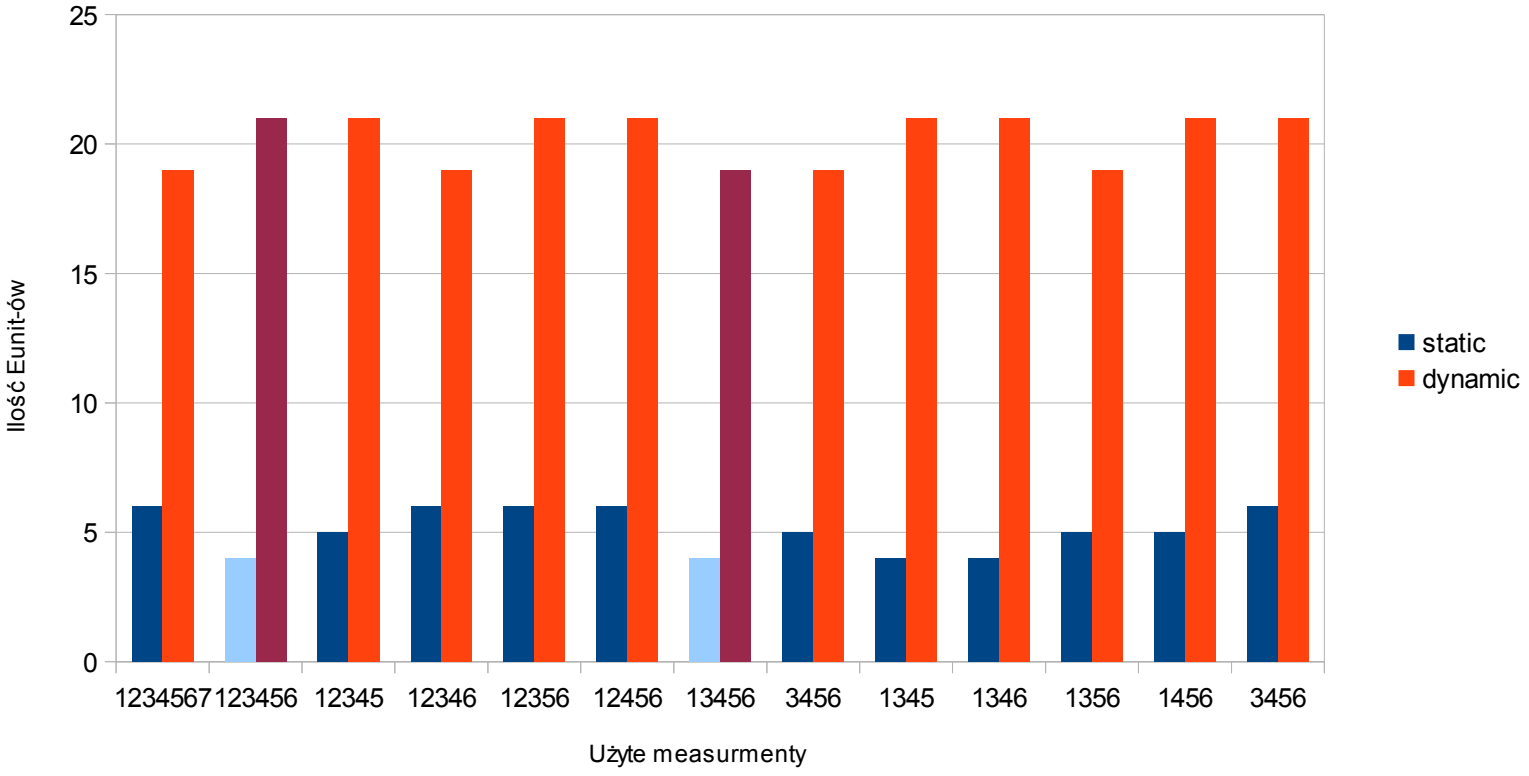
Na czerwono oznaczono wyniki dla benchmarka z statycznymi zamówieniami natomiast na niebiesko wyniki dla benchmarka z dynamicznymi zamówieniami.

Interpretacja wyników :

- przy wyborze wszystkich miar wyniki nie były zadawalające,
- po odrzuceniu miary MaxWaitTime wynik dla benchmarka statycznego o poprawił się o około 35 procent, stwierdzono, że należy pozbyć się tego measurmentu,
- przeprowadzano pomiary dla miar 1,2,3,4,5,6 odrzucając po kolei 6,5,4,3,2,
- przy odrzuceniu 2 miary (AverageMaxTimeWinSizeForAllCommissions) zauważono nieznaczną poprawę w stosunku do najlepszej dotychczasowej konfiguracji (1,2,3,4,5,6), poprawił się wynik dla benchmarka dynamicznego, wynik dla benchmarka statycznego pozostał prawie bez zmian,
- dla konfiguracji 1,3,4,5,6 próbowano pozbywać się miar, po kolei 6,5,4,3,1,
- w żadnym z przypadków wynik się nie poprawił, uznano, że najlepsza konfiguracja to 1,2,4,5,6.

Wyniki dla dystansu oraz liczby eunitów :

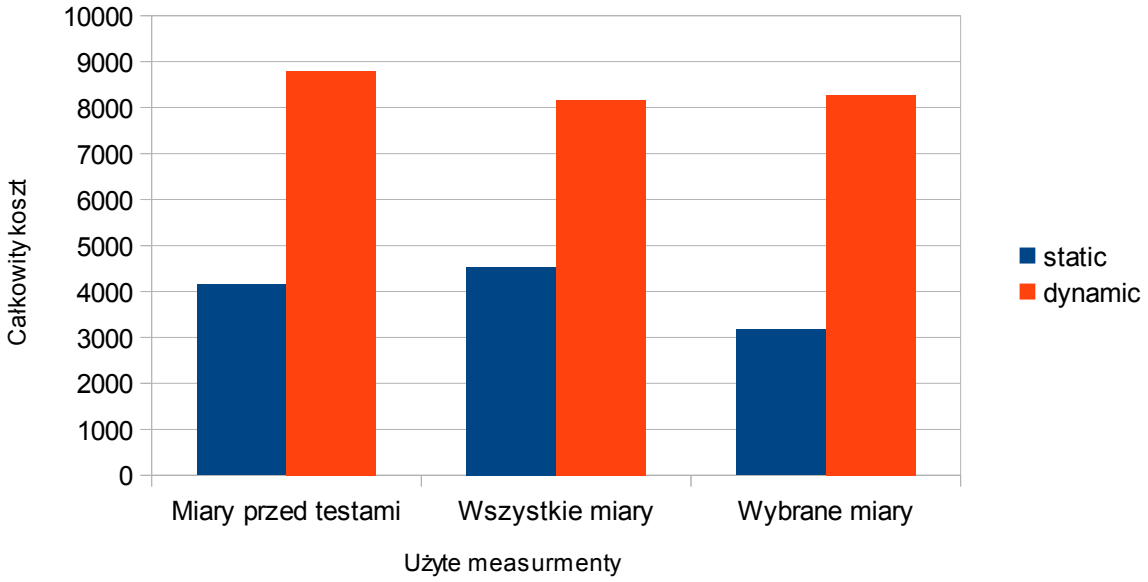


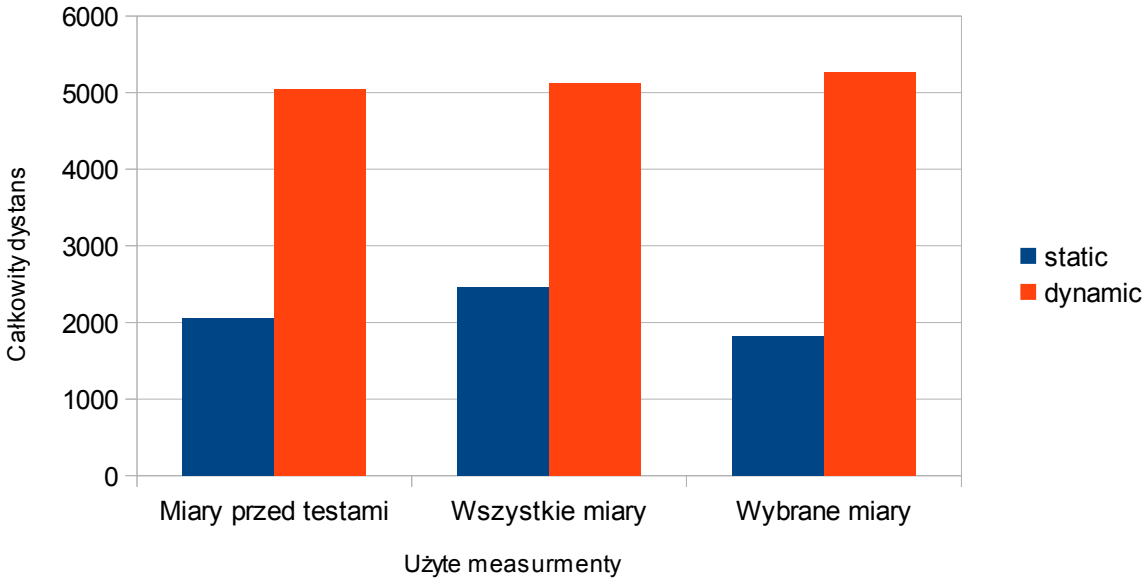


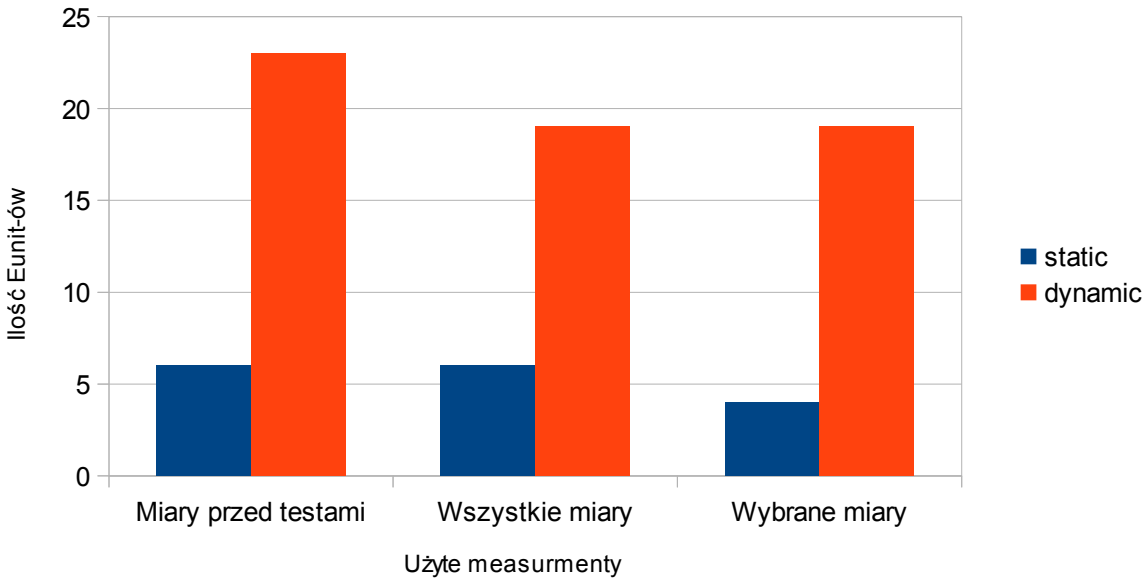
Wybrane miary :
1.AverageLoadFromAllCommissions
3.AverageMinTimeWinSizeForAllCommissions
4.AverageMinDistBetweenAllCommissions
5.PercentageOfDelayComs
6.WaitTime

Odrzucone miary :
2.AverageMaxTimeWinSizeForAllCommissions
7.MaxWaitTime

Porównanie dla najlepszego rozwiązania :







Powyższe wykresy pokazują, że poszukiwanie zestawu najlepszych miar było dobrym pomysłem oraz dało bardzo dobre rezultaty. W porównaniu do miar jakie używaliśmy wcześniej (AverageMinDistBetweenAllCommissions, WaitTime, AverageNumberOfComsWithinTimeWinOfAllCommissions) wynik jest lepszy zarówno dla benchmarka statycznego jak i dynamicznego, przy czym dla benchmarka statycznego uzyskaliśmy wynik lepszy o około 25% (całkowity koszt).

10. Testy powtarzalności wyników

10.1. Realizacja

Powtórzono 10 razy ten sam test dla przypadku statycznego i dynamicznego.

Wybrane benchmarki:

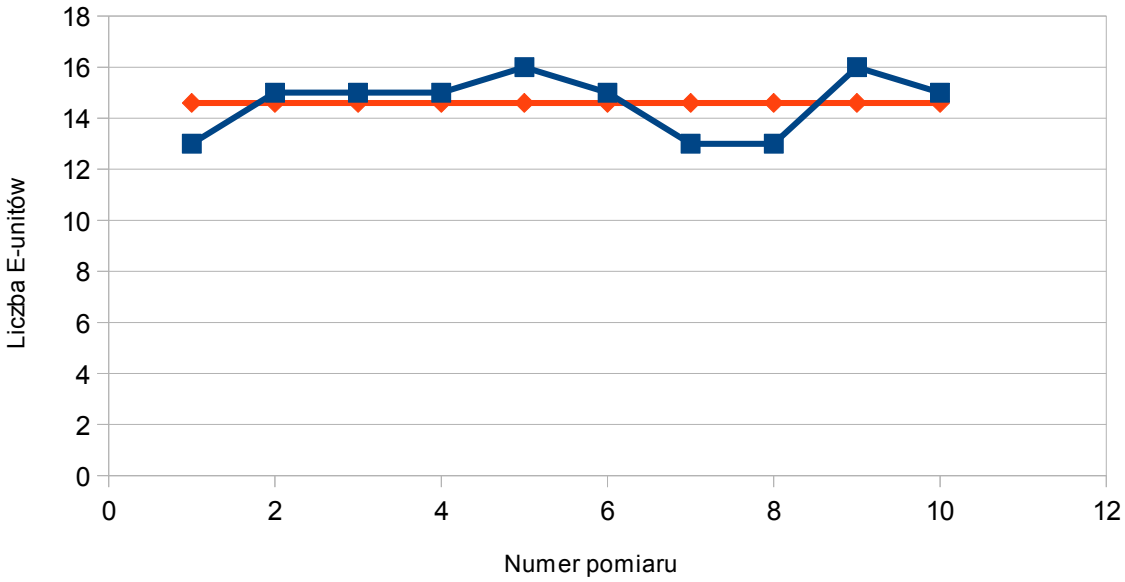
- static: 100, wąskie, lrc
- dynamic: A, 100, szerokie, lrc

10.2. Wyniki

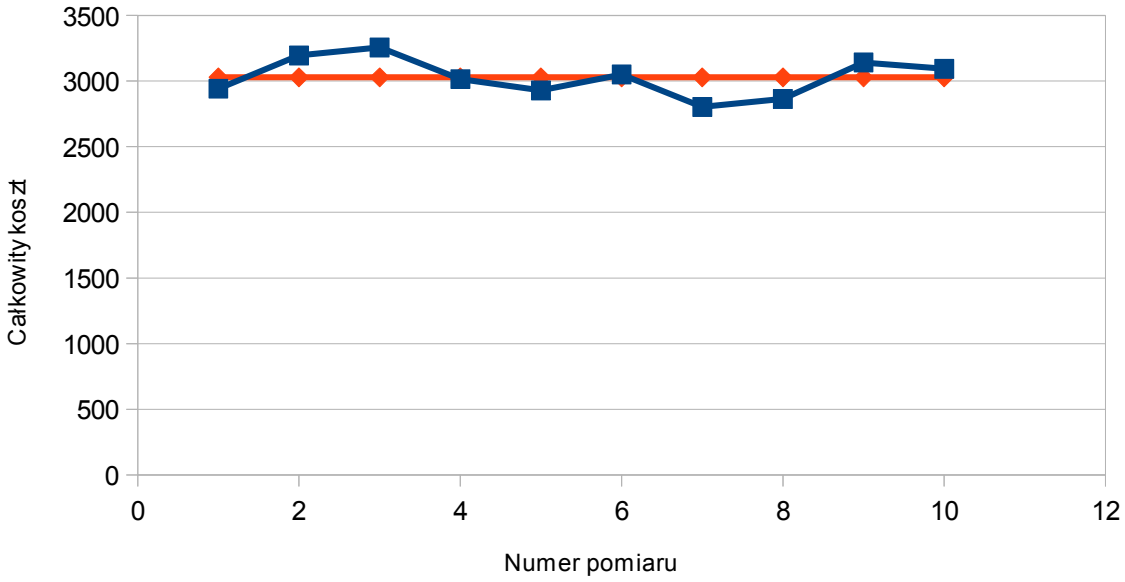
Na wykresach czerwoną linią jest zaznaczona wersja średnia, po wykresem zamieszczona jest wartość odchylenia.

Opis plików z wynikami :
Pliki z wynikami dla poniższych testów znajdują się w folderze „testyDeterminizm”. W środku mamy 2 foldery, „dynamic” dla problemu dynamicznego i „static” dla problemu statycznego. W środku znajdziemy pliki xls z wynikami dla kolejnych 10 pomiarów.

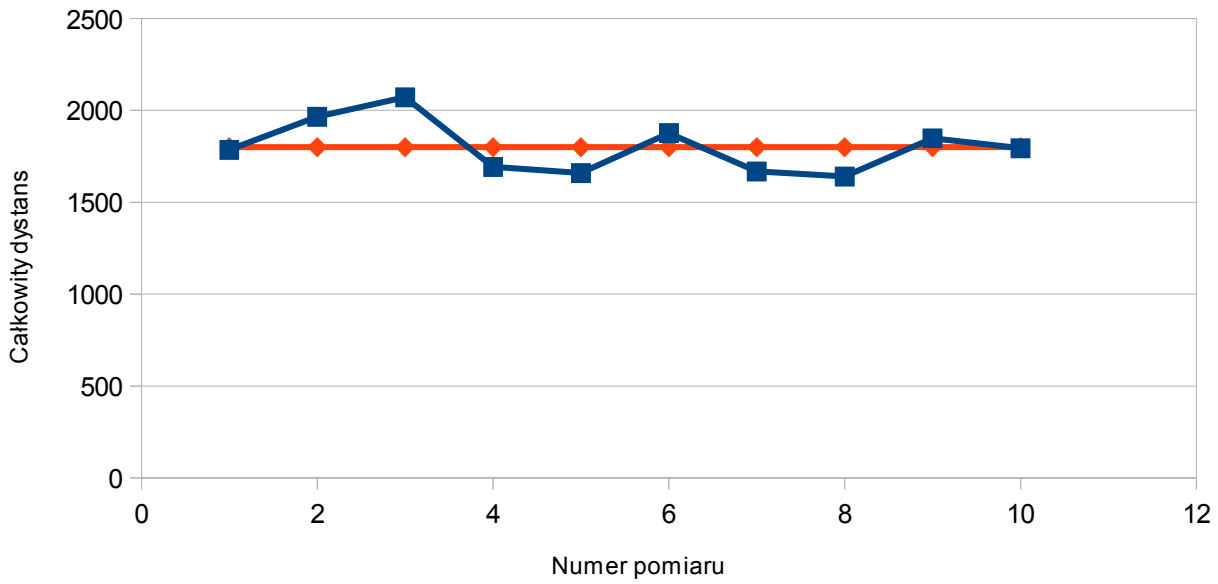
Benchmark statyczny.



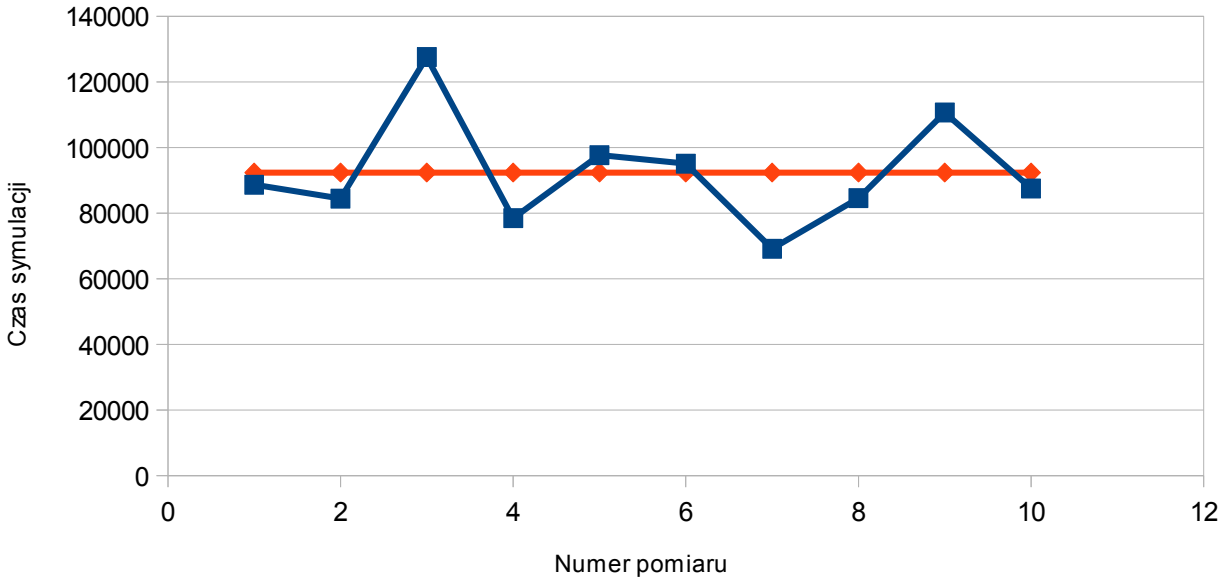
Odchylenie: 1,17



Odchylenie: 146,15

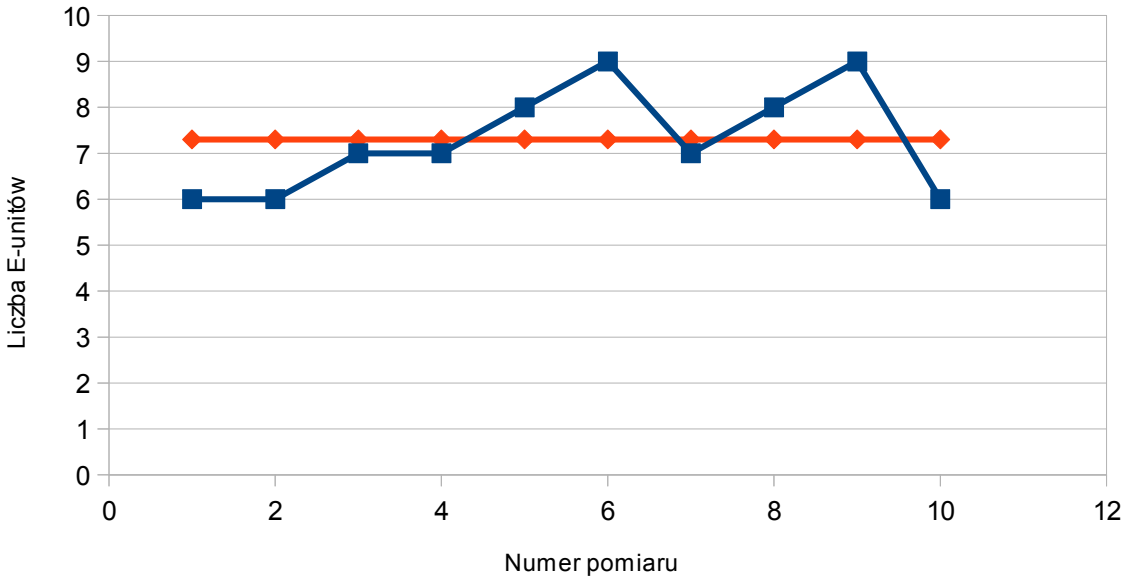


Odchylenie: 142,82

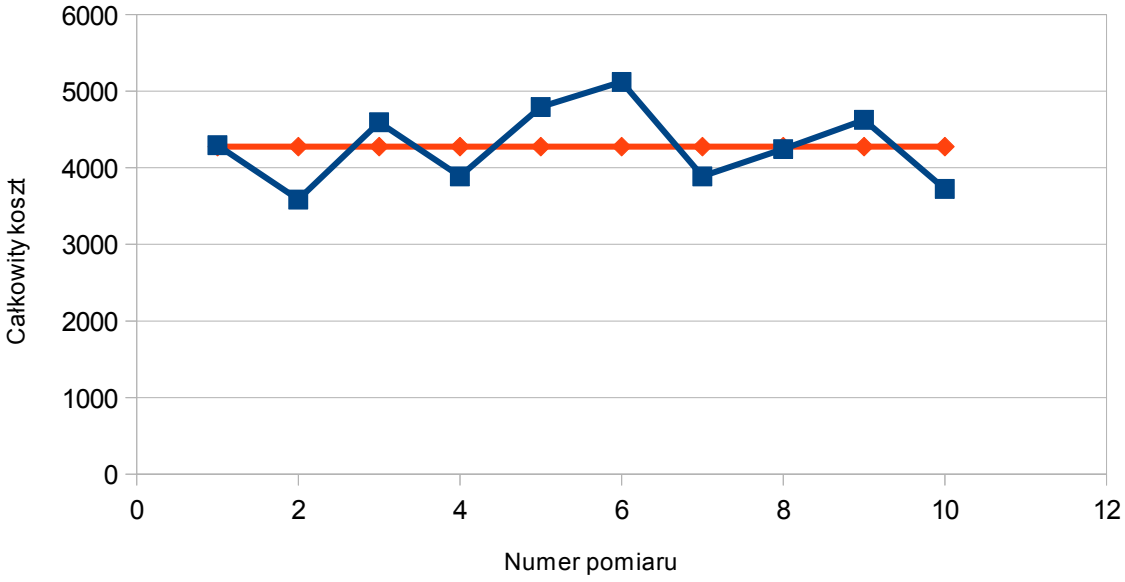


Odchylenie: 16676,21

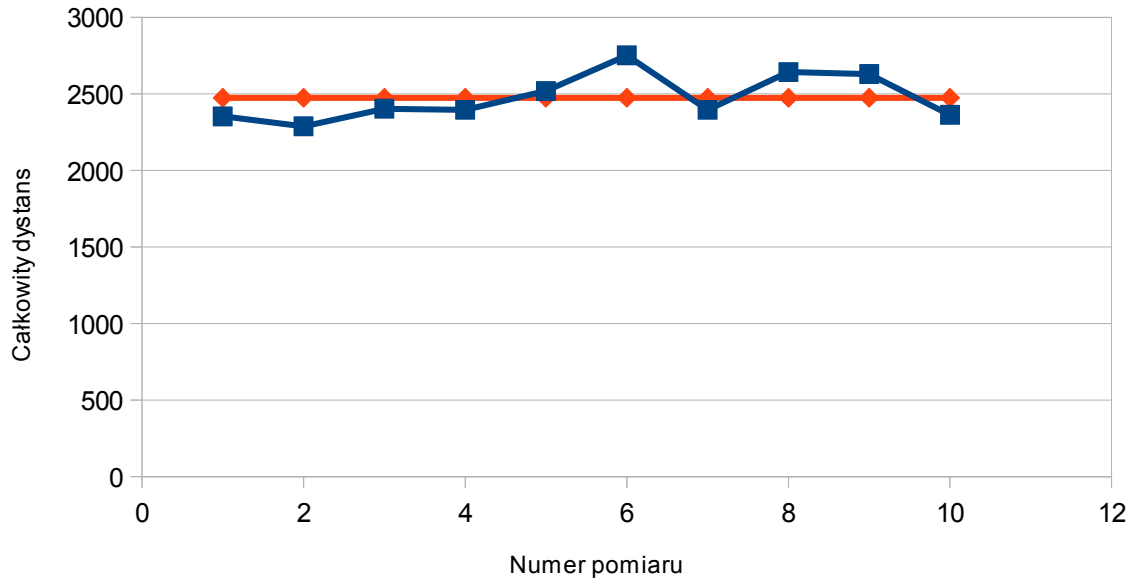
Benchmark dynamiczny



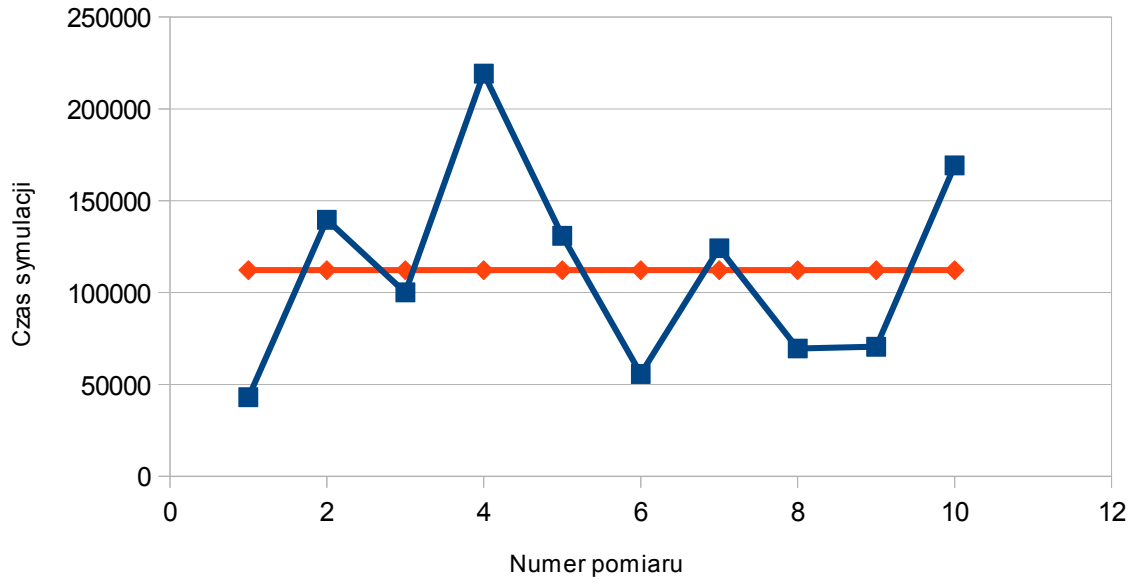
Odchylenie: 1,16



Odchylenie: 504,2



Odchylenie: 153,02



Odchylenie: 55395,15

11. Wnioski (Semestr I)

- Rezultaty lepsze lub bardzo zbliżone w porównaniu z poprzednim rozwiązaniem
- Czas symulacji nieznacznie większy – narzut uruchomienia platformy R
- Znacznie wygodniejsze użytkowanie – najważniejsza część konfiguracji tworzona jest automatycznie w trybie uczenia
- Użytkownik nie musi posiadać kompletnej wiedzy o działaniu o systemu – nie traci czasu na

opracowanie i zapisanie statycznego zbioru stanów
- Rozpoznanie stanu systemu jest gwarantowane – aktualny stan systemu zostanie przyporządkowany do istniejącego klastra, w poprzednim rozwiązaniu sami musimy zadbać aby statyczna konfiguracja obejmowała cały zbiór R (Real)
- Bardzo dobry model uczenia maszynowego – każda próba ingerencji w klastry wygenerowane w trybie uczenia dawała gorsze rezultaty
- Duże możliwości rozwojowe – algorytmy klasteringu, optymalna ilość klastrów, miary odległości punktu od klastra

12. Future work (Semestr I)

- Wymienny algorytm klasteringu – implementacja klastrowania hierarchicznego
- Dalsza automatyzacja, rozwój uczenia maszynowego – system sam decyduje których miar użyć na podstawie historii wartości
- Przenoszenie stanu tablicy akcji między trybami uczenia – w tej chwili po uruchomieniu trybu uczenia, tablica akcji jest zerowana, można zastosować rozwiązanie, w którym ustalamy metrykę i na jej podstawie liczymy odległości między starymi stanami i nowymi, jeśli odległość jest wystarczająco mała – przenosimy stan akcji ze starego stanu do nowego
- Próba implementacji uczenia maszynowego ukierunkowanego na konkretny cel np. minimalny koszt lub minimalny dystans realizacji zamówień

13. Podsumowanie pracy nad projektem (Semestr I)

Projekt został zrealizowany zgodnie ze stawianymi wymaganiami. Prace przebiegały zgodnie z harmonogramem i zostały ukończone na czas. Świadczy to o właściwej ocenie ryzyka oraz dobrym zarządzaniu projektem. Prawidłowe ustalenie priorytetów wymagań funkcjonalnych, zapewniły efektywną pracę zespołu developerskiego. Zostały zrealizowane wszystkie wymagania funkcjonalne. Przed zaprezentowaniem produktu zostały przeprowadzone liczne testy – w fazie początkowej jednostkowe, po naprawieniu drobnych błędów – wydajnościowe. Nie zostały wykryte żadne poważne problemy, a przeprowadzone testy jakościowe dały zadowalające wyniki. Uzyskano wysoką wydajność produktu i znacznie zwiększono wygodę użytkowania. Zaimplementowany moduł jest w pełni kompatybilny z resztą projektu i jest dobrze przygotowany do integracji z pozostałymi modułami, które mają zostać dodane do produktu. Moduł w dalszym ciągu wykazuje wiele możliwości rozwoju co daje nadzieję na dodatkowe zwiększenie jakości produktu.
Biorąc powyższe pod uwagę należy stwierdzić, że projekt zakończył się sukcesem.

14. Wnioski (Semestr II)

- Rezultaty dla drzew decyzyjnych porównywalne do rozwiązania, w którym użyto predykcji kmeans. Świadczy to o poprawności zaimplementowanego modułu
- Drzewo decyzyjne jest budowane tylko raz w czasie działania symulacji, wyszukiwanie w drzewie decyzyjnym jest prostą operacją. Dzięki temu czas symulacji nieznacznie się zmniejszył
- Dodano nowe opcje konfiguracji przez co system stał się jeszcze prostszy w użyciu oraz istnieje większa możliwość sterowania symulacją
- Poprzez poddział symulacji na etapy, nowe rozwiązanie zostało w pełni zintegrowane z istniejącym systemem
- Rozpoznanie stanu systemu jest gwarantowane - mamy pewność, że zbudowane drzewo będzie pełne, tzn. pokryje całą przestrzeń użytych miar,
- Wybór najlepszych miar poprawił znacząco wyniki dla benchmarków, dla których optymalizacja była przeprowadzana
- W celu uzyskania lepszych wyników symulacji dalsza praca powinna się skupić nad implementacją i/lub poprawieniem istniejących algorytmów wyboru tras

15. Future work (Semestr II)

- Zastosowanie klastrowania hierarchicznego – może zostać wykorzystane w pierwszym etapie algorytmu, do analizy skupień
- Automatyczny wybór miar wykorzystanych w symulacji – trudne do realizacji ale może przynieść dobre rezultaty
- poprawa innych części implementacji – ciągła poprawa algorytmów uczenia maszynowego nie zapewni dużo lepszych wyników, należy przyjrzeć się pozostałym algorytmom, sprawdzić ich poprawność oraz zaimplementować nowe

16. Podsumowanie pracy nad projektem (Semestr II)

Zrealizowany etap implementacji miał rozszerzyć moduł uczenia maszynowego o wykorzystanie drzew decyzyjnych. Poprzednia implementacja klasteringu kmeans nie została zmieniona – aplikacja została przygotowana na rozszerzenie. Nowy algorytm uczenia maszynowa został dodany jako opcja konfiguracyjna a cała symulacja podzielona na etapy, z których każdy ma wymienny algorytm. Uzyskano nową funkcjonalność oraz umożliwiony jeszcze łatwiejszy rozwój. Implementacja drzew decyzyjnych została przetestowana pod kątem poprawności oraz wydajności. Uzyskane rozwiązanie dało oczekiwane rezultaty. Należy podkreślić, że zarówno poprzednie jaki i nowe funkcje są dostępne dla użytkownika bez dogłębnej wiedzy o działaniu aplikacji – wszystko jest dostępne przez zewnętrzną konfigurację. Dodatkowo przeprowadzono testy wyboru najbardziej przydatnych w symulacji miar – dzięki temu użytkownik może mieć pewność, że wyniki działania będą zadowalające. Prace developerskie przebiegały równolegle z testowaniem oraz dokumentowaniem postępów dlatego nie pojawiły się żadne opóźnienia. Zostały zrealizowane wszystkie wymagania, projekt zakończył się powodzeniem.