Systemy Agentowe

Implementacja algorytmu opartego na miarach SNA do optymalizacji ruchu drogowego

Modyfikacje systemu Kraksim

Autorzy:
Maciej Kurek
Grzegorz Rajczyk

Prowadzący:
dr inż. Jarosław Koźlak

Zawartość

1. Wstęp

1.1. Cel

Podstawowym założeniem wprowadzanych zmian było umożliwienie wykorzystania algorytmów opartych na miarach SNA [2] do optymalizacji ruchu drogowego w systemie Kraksim. Dodatkowo chcieliśmy również uzyskać możliwość wizualizacji procesów bazujących na sieciach społecznych (grafy podziału, zmiany miar poszczególnych węzłów).

1.2. System Kraksim

Kraksim jest systemem umożliwiającym modelowanie ruchu drogowego. Do dyspozycji mamy takie elementy jak ulice z możliwością zdefiniowania liczby pasów z wyróżnieniem tych do jazdy na wprost oraz do skrętu w prawo i lewo, skrzyżowania, które tak jak w normalnym ruchu drogowym sterują ruchem drogowym za pomocą sygnalizacji świetlnych. Możemy również zdefiniować tzw. gateway'e, czyli miejsca, które pełnią podwójną rolę w modelu. Wszystkie samochody zarówno rozpoczynają tam swoją podróż jak i ją kończą. Dodatkowym atrybutem jest ilość „produkowanych" aut w jednostce czasu, co bezpośrednio przekłada się na natężenie ruchu na drogach.

Kraksim jest systemem jedynie symulującym i wizualizującym ruch na drogach. W celu stworzenia modelu miasta możemy się posłużyć systemem Kraksim City Designer, który pozawala nam na zamodelowanie większości wyżej opisanych elementów wchodzących w skład struktury miasta w systemie Kraksim.

2. Synchronizacja ruchu

2.1. Podział grafu na podgrafy

Do podziału grafu na podgrafy rozważamy implementację jednego z dwóch algorytmów. Pierwszy z nich to podział grafu na klastry. Wybieramy wierzchołki (skrzyżowania) o najwyższych wartościach miar obecnych już w systemie (tj. Page Rank lub Betweenness Centrality), które odpowiedzialne będą za sterowanie ruchem w podgrafie, jak i za negocjacje z innymi podgrafami co do ruchu samochodów pomiędzy nimi. Utworzenie podgrafów polegać będzie na dołożeniu do wyżej wymienionych głównych wierzchołków zlokalizowanych w bliskiej odległości skrzyżowań. Drugą opcją jest implementacja wielopoziomowego algorytmu podziału grafu zgodnie z opisem zawartym w referacie [7]. Wewnątrz wielopoziomowego algorytmu, po zmniejszeniu grafu wyjściowego, do podziału na podgrafy zastosowany zostanie wariant algorytmu Kernighana-Lina. Ten algorytm został wybrany, dlatego że zapewnia on minimalną ilość krawędzi łączących podgrafy, co zapewni nam duży poziom izolacji utworzonych podgrafów. Wysoki poziom izolacji podgrafów jest pożądany ze względu na konieczność wprowadzenia hierarchii wewnątrz każdego z podgrafów. Również przy małej ilości krawędzi (dróg) łączących podgrafy łatwiej będzie wprowadzić synchronizację świateł pomiędzy podgrafami. Algorytmy służące do podziału grafu muszą zostać zmodyfikowane w taki sposób aby brały pod uwagę zmieniające się wartości miar węzłów, a co za tym idzie dzieliły graf w różny sposób, zależny od natężenia ruchu w danym momencie.

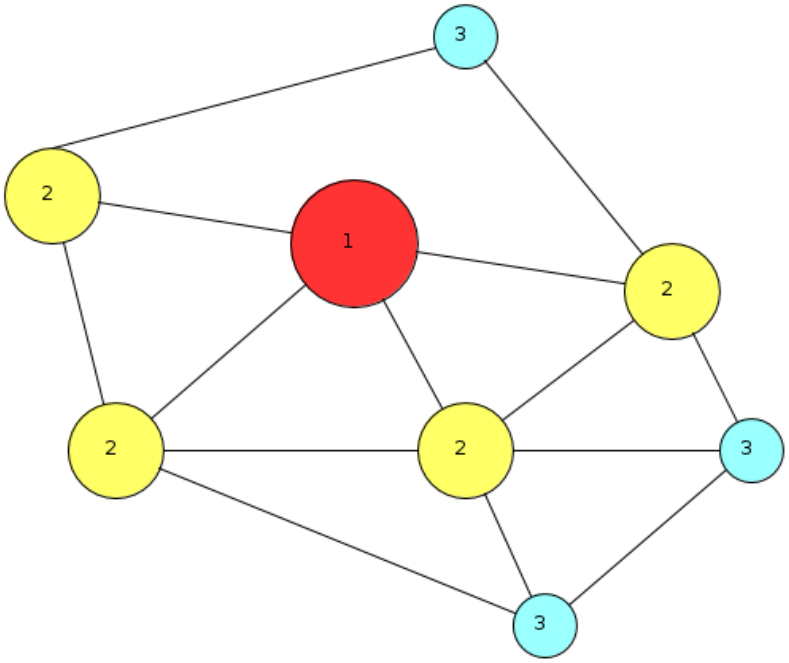
2.2. Klasteryzacja grafu

Ostatecznie jako metodę podziału grafu na podgrafy wybraliśmy klasteryzację wykorzystującą metodę k-mens. Klasteryzacja grafu przebiega dwu-etapowo. Najpierw wybieramy zadaną ilość wierzchołków o najwyższych wartościach miar (a zarazem największym natężeniu ruchu) jako środki naszych klastrów. Następnie dla każdego skrzyżowania sprawdzamy, do którego skrzyżowania głównego (środka klastra) ma najbliżej. Po dokonaniu sprawdzenia, skrzyżowanie jest przypisywane do odpowiedniego klastra. W ten sposób otrzymujemy zadaną ilość klastrów, w których wszystkie skrzyżowania mają najbliżej do skrzyżowania głównego (środka klastra).

2.3. Zarządzanie ruchem wewnątrz podgrafów

2.3.1. Wprowadzenie hierarchii

Wewnątrz każdego podgrafu wprowadzony zostanie hierarchiczny podział węzłów. Struktura hierarchiczna zostanie wprowadzona na podstawie obliczonej wartości miar węzłów, odwzorowując w ten sposób wielkość skrzyżowań oraz aktualne natężenie ruchu. Ilość poziomów hierarchii będzie wybierana podczas konfiguracji symulacji (w późniejszych wersjach systemu być może będzie można zmieniać ilość poziomów dynamicznie, podczas działania symulacji).

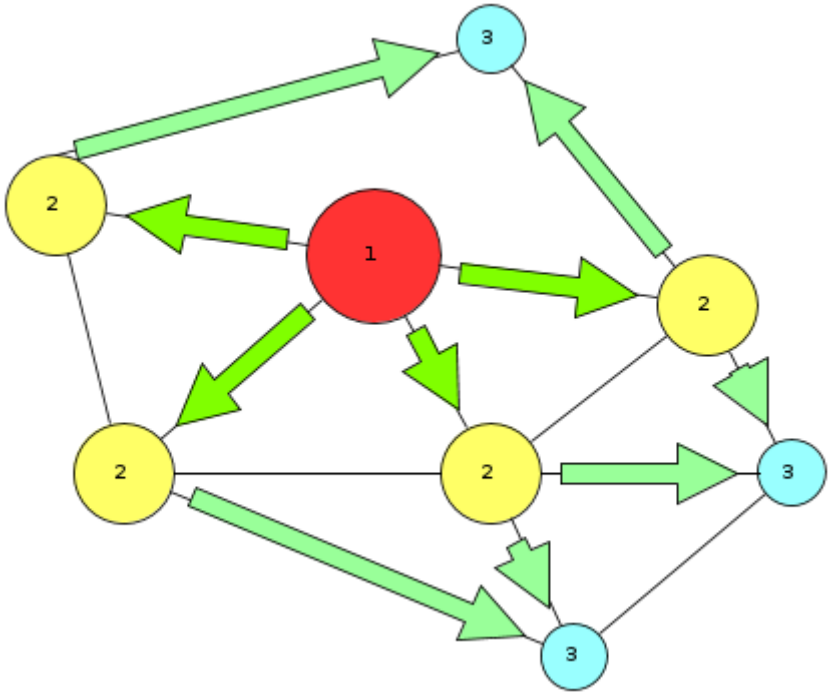


Rysunek 1 - Podgraf z wprowadzoną hierarchią ważności. Numery przedstawiają pozycję w hierarchii, od najważniejszego węzła do najmniej ważnego.

Wprowadzanie do podgrafu hierarchii może mieć miejsce podczas podziału grafu na podgrafy, przy zastosowaniu algorytmu wielopoziomowego. Utworzenie struktury hierarchicznej można połączyć z etapem kurczenia (zmniejszania) grafu. Jeżeli natomiast wybrany zostanie algorytm dzielący graf na klastry, hierarchia wierzchołków w podgrafach będzie zależała tylko i wyłącznie od obliczonych wartości miar skrzyżowań.

2.3.2. Wykorzystanie hierarchii do synchronizacji świateł

Stworzenie hierarchicznej struktury jest potrzebne aby można było przeprowadzić synchronizację świateł. Światła na skrzyżowaniach na niższym poziomie w hierarchii będą ustawiane na podstawie ustawień skrzyżowań znajdujących się wyżej w hierarchii, tak aby zapewnić jak najmniejsze natężenie ruchu pomiędzy skrzyżowaniami ważniejszymi a mniej ważnymi. Światła na skrzyżowaniach będą ustawiane kolejno, zgodnie z pozycją w hierarchii a informacja dotycząca synchronizacji świateł będzie rozpropagowywana w dół hierarchii, od węzłów najważniejszych do najmniej ważnych.



Rysunek 2 - Propagacja informacji dotyczących synchronizacji świateł w podgrafie z hierarchiczną strukturą

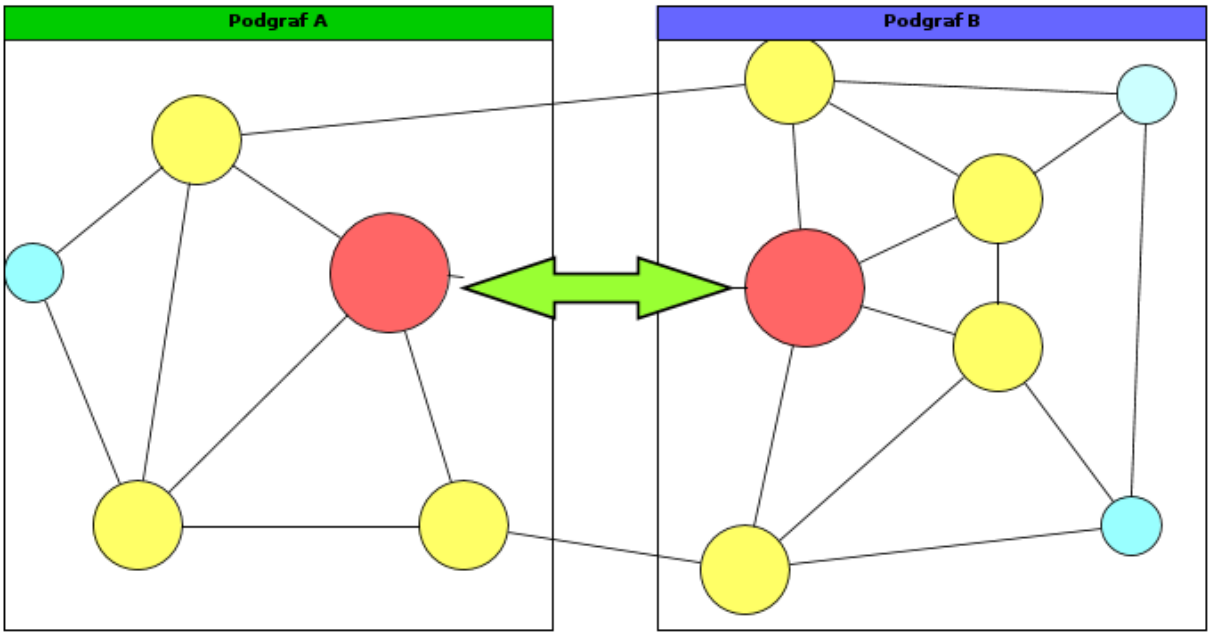
Na powyższym rysunku (Rysunek 2) przedstawiony został sposób propagacji informacji dotyczących ustawień świateł. Dla każdego podgrafu światła będą ustawiane w następujący sposób:

- Ustawienie świateł na skrzyżowaniu poziomu 1 (w oparciu interakcje i innymi podgrafami).
- Propagacja ustawienia świateł na skrzyżowaniu 1 do powiązanych skrzyżowań poziomu 2.
- Ustawienie świateł na skrzyżowaniach poziomu 2 (w oparciu o otrzymaną od skrzyżowania poziomu 1 informację).
- Propagacja ustawień świateł na skrzyżowaniach poziomu 2 do powiązanych skrzyżowań poziomu 3.
- Ustawienie świateł na skrzyżowaniach poziomu 3 (w oparciu o otrzymaną od skrzyżowań poziomu 2 informację).

2.4. Interakcje pomiędzy podgrafami

Podział grafu na podgrafy nie zostaje wprowadzony jedynie po to aby usprawnić synchronizację świateł wewnątrz podgrafu. Podgrafy będą komunikować się ze sobą w celu synchronizacji świateł na najważniejszych skrzyżowaniach. Wymiana informacji powinna występować pomiędzy

najważniejszymi skrzyżowaniami (węzłami będącymi najwyżej w hierarchii) podgrafów w celu takiego ustawienia świateł, aby natężenie ruchu (korki) pomiędzy podgrafami było jak najmniejsze.



Rysunek 3 - Przesyłanie informacji pomiędzy podgrafami. Wymiana informacji dotyczącej synchronizacji świateł odbywa się pomiędzy najważniejszymi węzłami w podgrafach.

2.5. Sposób negocjacji pomiędzy podgrafami

Aby można było dokonać efektywnej synchronizacji świateł wewnątrz podgrafów, muszą one wpierw wynegocjować optymalne zestawienie świateł na głównych trasach łączących węzły o najwyższych wartościach miar. To one muszą ustalić, które trasy mają największy priorytet, i muszą mieć bezwzględnie zsynchronizowane światła, tak by zapobiec tworzeniu się zatorów na tych strategicznych trasach. W niektórych przypadkach być może niemożliwe będzie udrożnienie wszystkich najważniejszych dróg, więc główne skrzyżowania będą negocjować między sobą najbardziej optymalne ustawienie świateł, biorąc pod uwagę chwilowe natężenia na ulicach. Niektóre węzły będą zatem musiały być chwilowo „pokrzywdzone", tzn. trasy przez nie przechodzące nie będą do końca zsynchronizowane przez co będą się na nich tworzyć zatory. Negocjacje powinny być prowadzone w taki sposób aby długoterminowe średnie poziomy zadowolenia każdego z węzłów były zbliżone. Innymi słowy korki nie mogą tworzyć się ciągle w jednym miejscu. Dopiero po osiągnięciu porozumienia pomiędzy głównymi węzłami grafu, można dokonywać synchronizacji opisanej w punkcie 1.2.2 na poziomie podgrafów.

3. Synchronizacja świateł

3.1. Mechanizm negocjacji oraz propagacji ustawień świateł.

1. Na głównych skrzyżowaniach (skrzyżowania o najwyższych wagach – środki klastrów), regulują czas zmiany świateł w taki sposób aby wydłużyć czas palenia się zielonego światła na dróg wjazdowych o dużym natężeniu ruchu (większe zakorkowanie) jednocześnie skracając ten czas dla dróg o mniejszym natężeniu ruchu.
2. Gdy każde ze skrzyżowań głównych ustawi swoje światła następuje wymiana informacji o ustawieniach świateł między skrzyżowaniami głównymi.

3. Skrzyżowania główne na podstawie wymienionych informacji korygują ustawienia świateł w następujący sposób:
    a. Jeśli otrzymane informacje pochodzą od skrzyżowania głównego będącego w bezpośrednim kontakcie (połączone drogą, bez żadnych innych skrzyżowań po drodze) z danym skrzyżowaniem, ustawienia świateł są korygowane tak aby dostosować się do ustawień sąsiada (sposób korygowania opisany zostanie poniżej).
    b. W przeciwnym przypadku otrzymane informacje są ignorowane.
4. Zależnie od efektów (badania sprawdzone w trakcie testów) punkt 3 może być wykonywany jednokrotnie lub też tak długo aż na żadnym skrzyżowaniu nie zostanie dokonana korekta. Zastosowany wariant zostanie wybrany w zależności od efektywności oraz czasu potrzebnego na wykonanie takich operacji.
5. Gdy na skrzyżowaniach głównych zostaną już ustalone światła informacja o ustawieniu świateł zostanie rozpropagowana do węzłów będących niżej w hierarchii w obrębie danego klastra. Propagacja informacji odbywać się będzie w kolejnych iteracjach, tak aby zapobiec zmiany świateł na podstawie informacji otrzymanych od węzłów będących niżej w hierarchii (Jeśli na danym skrzyżowaniu zostały skonfigurowane światła w danej iteracji, to w następnej iteracji może on otrzymać konfigurację świateł od węzła będącego niżej w hierarchii – taka informacja zostanie zignorowana).
6. Gdy we wszystkich węzłach w klastrach zostanie poprawiona konfiguracja świateł, kończymy algorytm.

3.2. Mechanizm korekcji świateł

Jeśli skrzyżowanie otrzyma informacje że na sąsiednim skrzyżowaniu, czas palenia się zielonego światła został wydłużony dla drogi łączącej dane skrzyżowanie z sąsiednim to skrzyżowanie powinno również wydłużyć czas palenia się zielonego światła dla tej drogi. Jeśli czas palenia się zielonego światła został skrócony, skrzyżowanie może również skrócić czas palenia się zielonego światła aby umożliwić wydłużenie tego czasu dla innych świateł na skrzyżowaniu. Dzięki takiemu mechanizmowi unikniemy sytuacji w której korki będą przenoszone z jednego skrzyżowania na sąsiadujące. Jeśli dojdzie do sytuacji że dwa (lub więcej) sąsiadujące skrzyżowania zwiększyły czas palenia się zielonego światła (dla dwóch różnych dróg wjazdowych na dane skrzyżowanie) krzyżowanie musi tak skonfigurować światła aby możliwie jak najlepiej dostosować synchronizację świateł do sąsiadów. W ten sposób tworzenie się korków na danym skrzyżowaniu zostanie maksymalnie zminimalizowane.

3.3. Mechanizm konfiguracji świateł – wersja uproszczona

Jeśli zrealizowanie standardowej wersji okaże się niemożliwe (z powodów czasowych lub z powodu dużej złożoności problemu) lub też nieefektywne, przygotowana została wersja uproszczona mechanizmu konfiguracji świateł oraz optymalizacji ruchu. Uproszczona wersja polegać będzie na odpowiednim ustawieniu świateł jedynie na skrzyżowaniach głównych oraz na wymianie informacji między nimi. Taki mechanizm pozwoli zminimalizować natężenie ruchu na najbardziej zakorkowanych skrzyżowaniach. Stosując taki mechanizm spowodujemy wystąpienie problemu, polegającego na tym iż korki będą przenoszone z najbardziej zakorkowanych skrzyżowań na skrzyżowania sąsiadujące z nimi. Jednak taka sytuacja nie jest krytyczna, gdyż w następnym etapie, skrzyżowania te zostaną oznaczone (poprzez miary oraz klastrowanie) jako skrzyżowania główne. Powtórnie realizując algorytm konfiguracji świateł znów uzyskamy przeniesienie korków. Dzięki temu uzyskamy rozładowywanie się korków na najbardziej zatłoczonych skrzyżowaniach kosztem przeniesienia ich na sąsiadujące skrzyżowania. Taka sytuacja poprawi jednak ogólne zakorkowanie

miasta jako że korki nie będą rosły w jednym miejscu tylko będą rozkładać się na sąsiednie skrzyżowania.

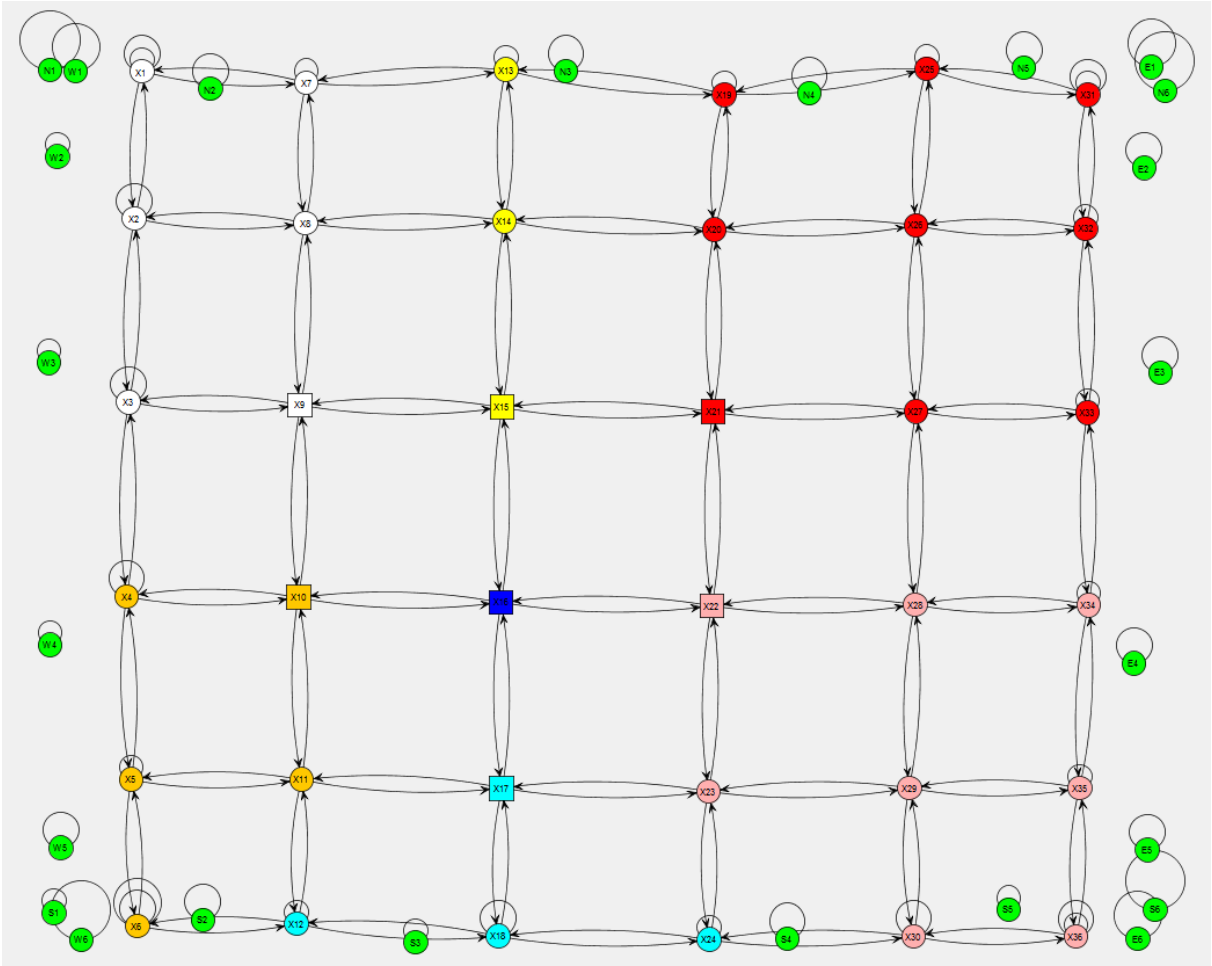
3.4. Synchronizacja świateł – zaimplementowany algorytm

Zaimplementowany przez nas algorytm jest połączeniem algorytmu uproszczonego (3.3) oraz podstawowego algorytmu (3.1). Pierwszym etapem jest konfiguracja świateł przez główne skrzyżowania, a następnie wymiana informacji pomiędzy głównymi skrzyżowaniami. Następnie, gdy główne skrzyżowania skorygują swoje ustawienia na podstawie wymienionych ze sobą informacji, propagują informacje dotyczące zmian konfiguracji świateł do węzłów wewnątrz swojego klastra. Węzły podrzędne, po otrzymaniu informacji od skrzyżowania głównego przystępują do konfiguracji własnych świateł. Jeśli węzeł podrzędny znajduje się w promieniu 2 skrzyżowań od skrzyżowania głównego, zmienia konfigurację swoich świateł na podstawie otrzymanych informacji. W przeciwnym przypadku skrzyżowanie nie dokonuje żadnych zmian w swojej konfiguracji.

Podczas testów sprawdziliśmy efektywność działania zarówno algorytmu uproszczonego jak i algorytmu będącego połączeniem algorytmu uproszczonego oraz podstawowego (tak jak to zostało opisane powyżej).

4. Wizualizacja grafu

Dodano widok reprezentacji grafu przy użyciu biblioteki JUNG. Graf ten obrazuje podział miasta na podgrafy. Poszczególne węzły mają nazwy analogiczne do skrzyżowań, które reprezentują. Rozmieszczenie węzłów w grafie ma w miarę możliwości naśladować oryginalne rozmieszczenie skrzyżowań w mieście. Poszczególne kolory węzłów reprezentują podział grafu na klastry na podstawie wyliczanych miar. Węzły kwadratowe reprezentują główne węzły w klastrze, natomiast węzły o okrągłym kształcie są podrzędnymi skrzyżowaniami danego klastra. Okrągłe węzły w grafie reprezentują skrzyżowania typu Gateway. Podczas integracji z nową wersją systemu, z nieznanych przez nas przyczyn węzły te nie są połączone ze skrzyżowaniami a połączenie to jest reprezentowane przez zapętlenia. Jako że nie udało nam się odnaleźć przyczyny takiego zachowania węzłów w nowej wersji systemu nie udało nam się poprawić tego błędu.



Rysunek 4 - Wizualizacja grafu, wraz z podziałem na klastry, węzły główne oraz podrzędne dla mapy o kształcie siatki

5. Testy

5.1. Opis testów

Przeprowadzone przez nas testy miały na celu sprawdzenie efektywności zaimplementowanego przez nas algorytmu optymalizacji ruchu drogowego, oraz porównanie go z istniejącym algorytmem zaimplementowanym w systemie. Wszystkie testy zostały przeprowadzone przy użyciu miar centralności PageRank (wybranej jako najlepiej odzwierciedlającą ważność skrzyżowań na podstawie ich rozmiarów oraz aktualnego natężenia ruchu, co zostało opisane w poprzednio dostarczonym sprawozdaniu). Testy zostały przeprowadzone dla dwóch rodzajów map:

- Siatka
- Mapa Krakowa – duża.

Użyliśmy trzech różnych algorytmów synchronizacji ruchu:

- Algorytm istniejący w systemie
- Uproszczona wersja naszego algorytmu (3.3)
- Rozszerzona wersja naszego algorytmu (3.4)

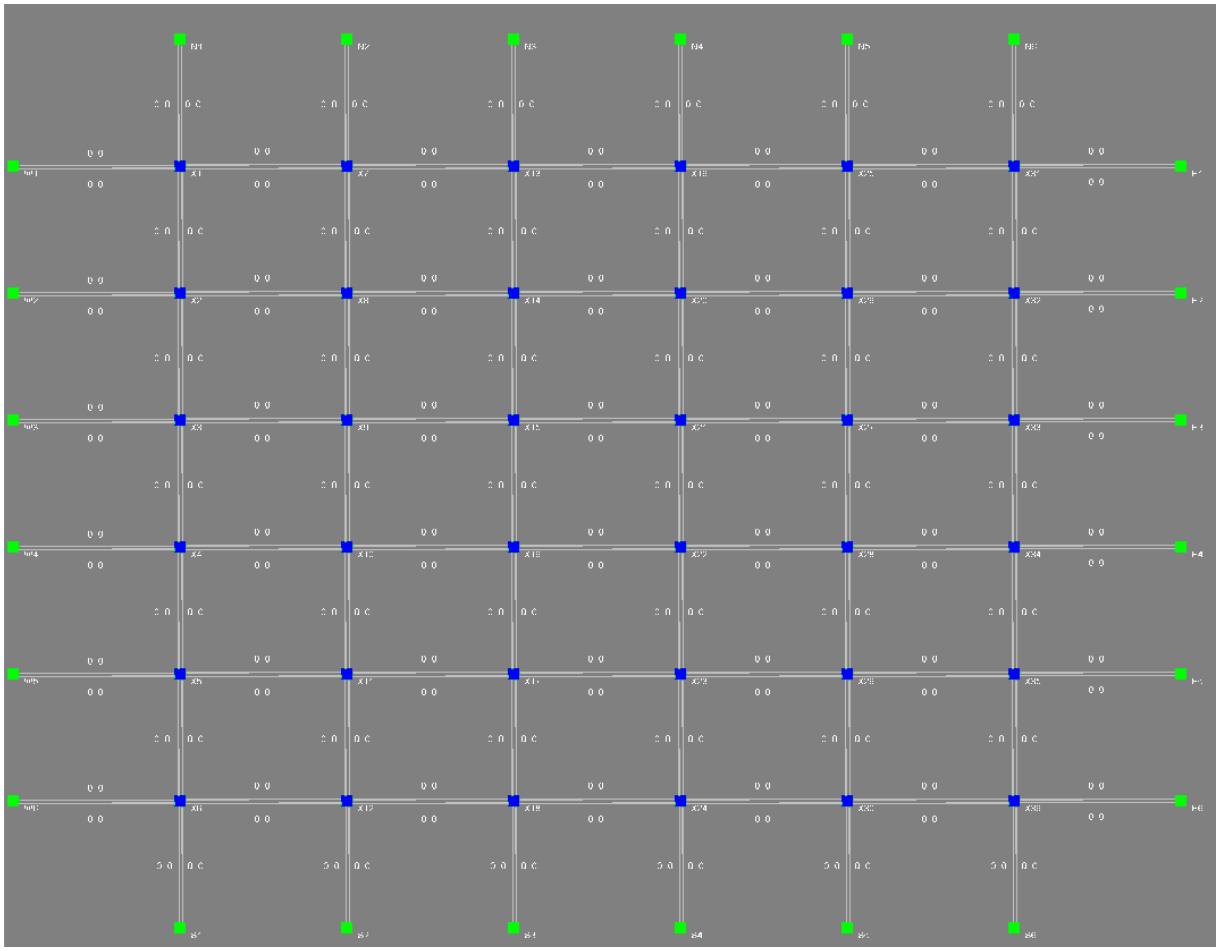
Wyniki wszystkich testów znajdują się w załączonym arkuszu kalkulacyjnym wraz z pojedynczymi wykresami.

5.2. Testy efektywnościowe – porównanie

5.2.1. Mapa – siatka

Poniższe wyniki testów odnoszą się do symulacji przeprowadzonych na mapie – siatce przedstawionej poniżej. Konfiguracja zastosowana do poniższych symulacji to:

- Mapa – 6x6map.xml
- Organizacja ruchu – 6x6_240000_800x.xml



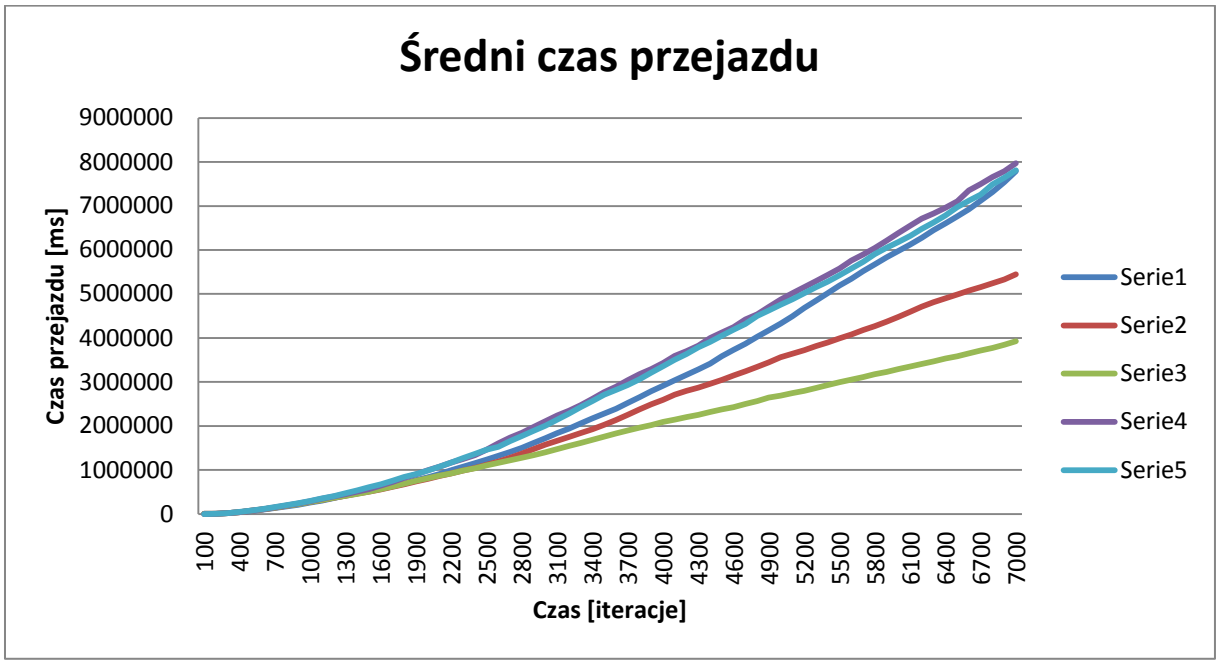
Rysunek 5 - Mapa w kształcie siatki

5.2.1.1. Opis danych znajdujących się na wykresie

- Serie 1 – Dane zebrane przy zastosowaniu algorytmu podstawowego, przy podziale na 4 klastry oraz przy użyciu algorytmu SOTL
- Serie 2 - Dane zebrane przy zastosowaniu algorytmu rozszerzonego, przy podziale na 4 klastry oraz przy użyciu algorytmu SOTL
- Serie3 – Dane zebrane przy zastosowaniu starego algorytmu z algorytmem SOTL
- Serie 4 - Dane zebrane przy zastosowaniu starego algorytmu z algorytmem RL
- Serie 5 - Dane zebrane przy zastosowaniu algorytmu rozszerzonego, przy podziale na 4 klastry oraz przy użyciu algorytmu RL

5.2.1.2. *Średnie czasy przejazdów*

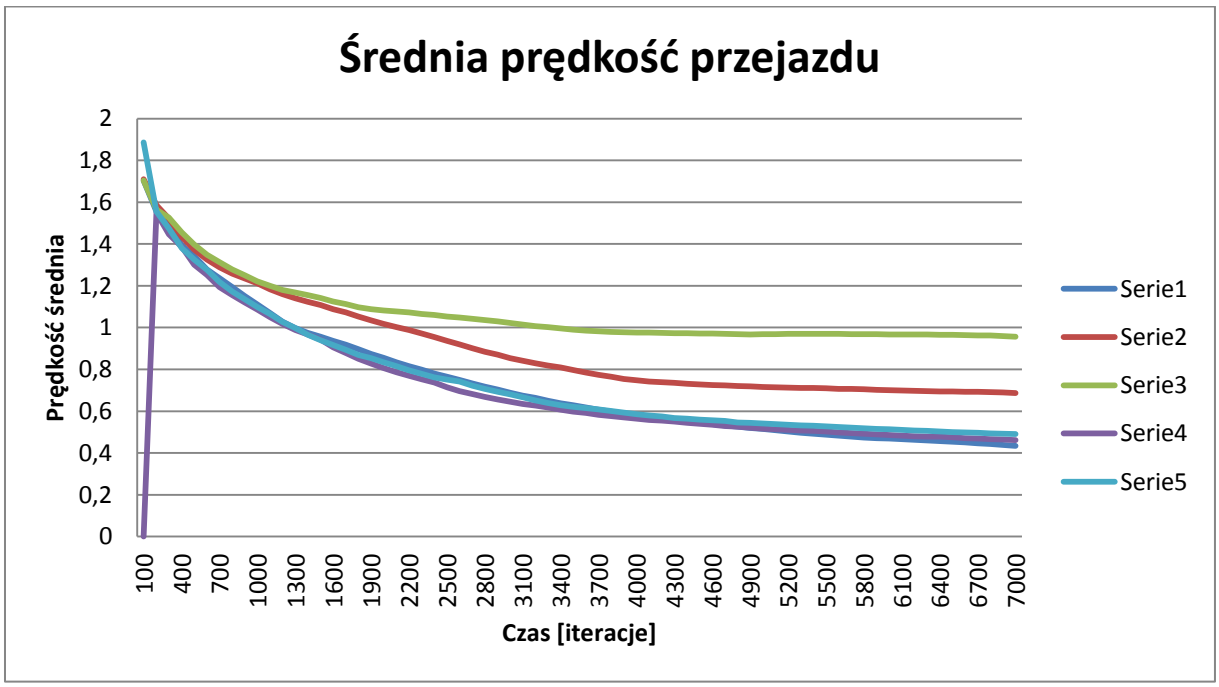
Poniższy wykres prezentuje zestawienie średnich czasów przejazdów pojazdów, mierzony po każdych 100 iteracjach.



Wykres 1- Średni czas przejazdów dla symulacji na mapie w kształcie siatki

5.2.1.3. *Średnie prędkości przejazdów*

Poniższy wykres prezentuje zestawienie średnich prędkości pojazdów, mierzony po każdych 100 iteracjach.

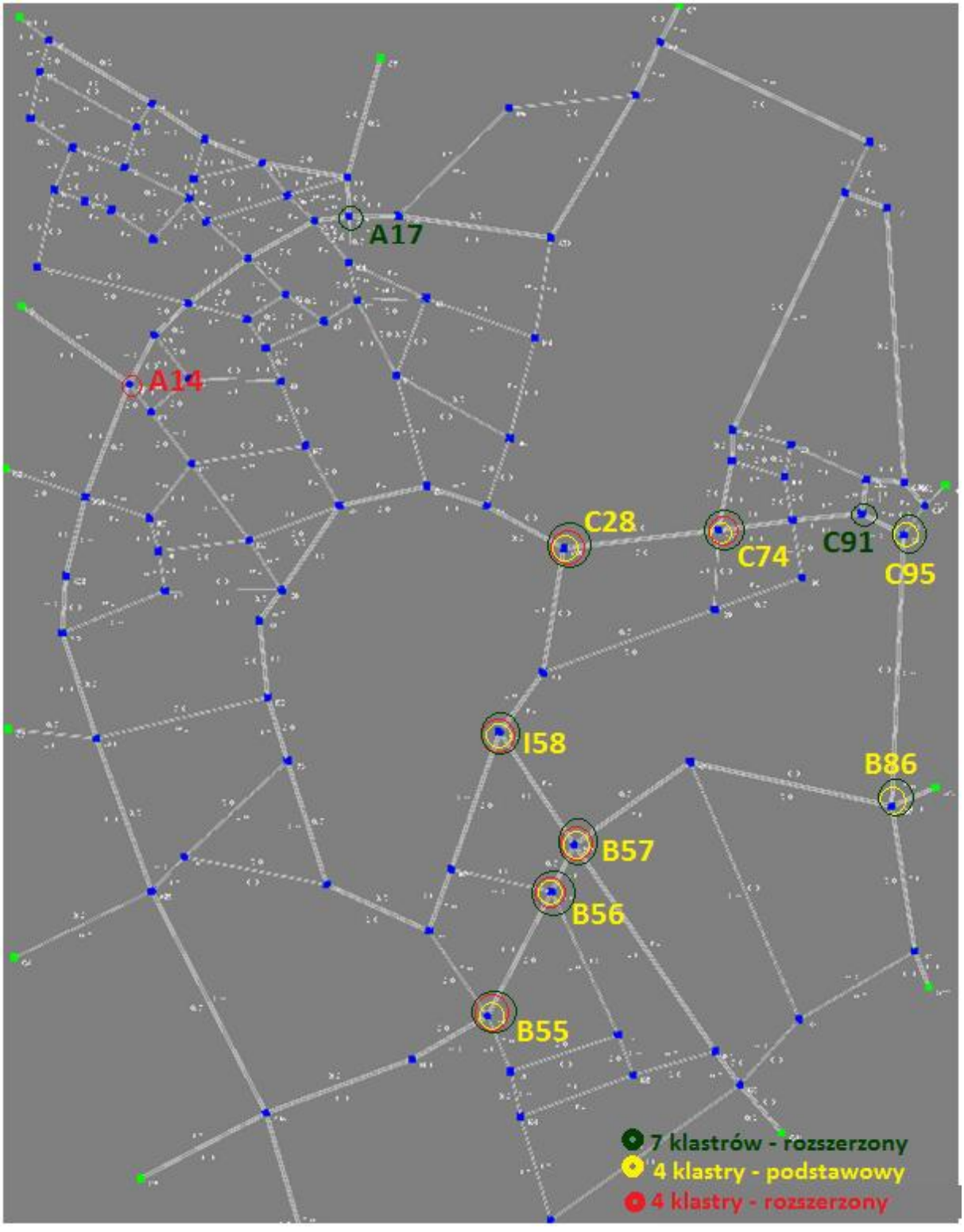


Wykres 2 - Średnie prędkości pojazdów dla symulacji na mapie w kształcie siatki

5.2.2. Mapa Krakowa – duża

Poniższe wyniki testów odnoszą się do symulacji przeprowadzonych na dużej mapie Krakowa przedstawionej poniżej. Konfiguracja zastosowana do poniższych symulacji to:

- Mapa – krakow_duzy.xml
- Organizacja ruchu - kra_240000_31x.xml



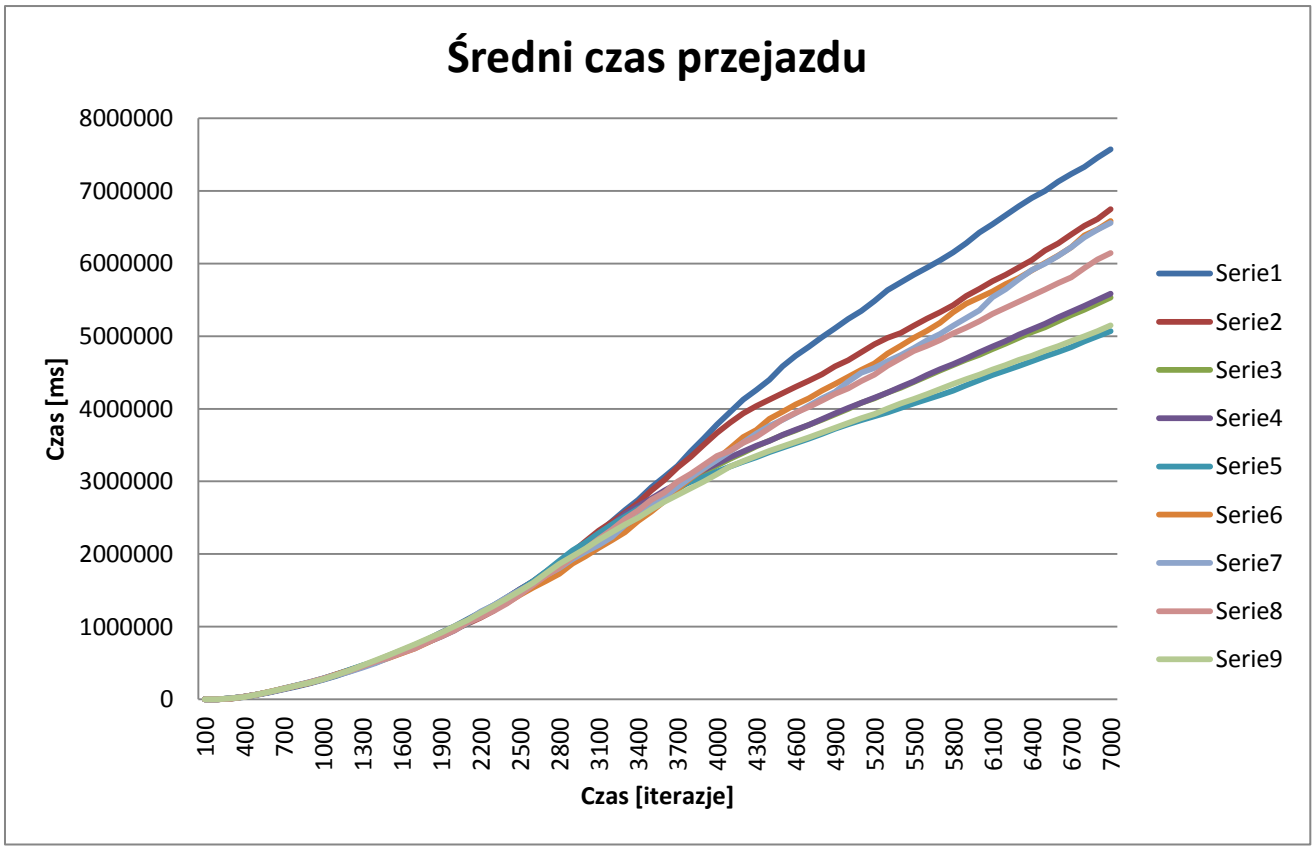
Rysunek 6 - Mapa Krakowa – duża

5.2.2.1. Opis danych znajdujących się na wykresie

- Serie 1 - Dane zebrane przy zastosowaniu algorytmu podstawowego, przy podziale na 4 klastry oraz przy użyciu algorytmu SOTL
- Serie 2 - Dane zebrane przy zastosowaniu algorytmu podstawowego, przy podziale na 10 klastry oraz przy użyciu algorytmu SOTL
- Serie 3 - Dane zebrane przy zastosowaniu algorytmu rozszezrzonego, przy podziale na 4 klastry oraz przy użyciu algorytmu SOTL
- Serie 4 - Dane zebrane przy zastosowaniu algorytmu rozszerzonego, przy podziale na 7 klastry oraz przy użyciu algorytmu SOTL
- Serie 5 – Dane zebrane przy zastosowaniu starego algorytmu z algorytmem SOTL
- Serie 6 - Dane zebrane przy zastosowaniu starego algorytmu z algorytmem RL
- Serie 7 - Dane zebrane przy zastosowaniu algorytmu rozszerzonego, przy podziale na 4 klastry oraz przy użyciu algorytmu RL
- Serie 8 - Dane zebrane przy zastosowaniu algorytmu rozszerzonego, przy podziale na 7 klastry oraz przy użyciu algorytmu RL
- Serie 9 - Dane zebrane przy zastosowaniu algorytmu rozszerzonego, przy podziale na 6 klastry oraz przy użyciu algorytmu SOTL

5.2.2.2. Średnie czasy przejazdów

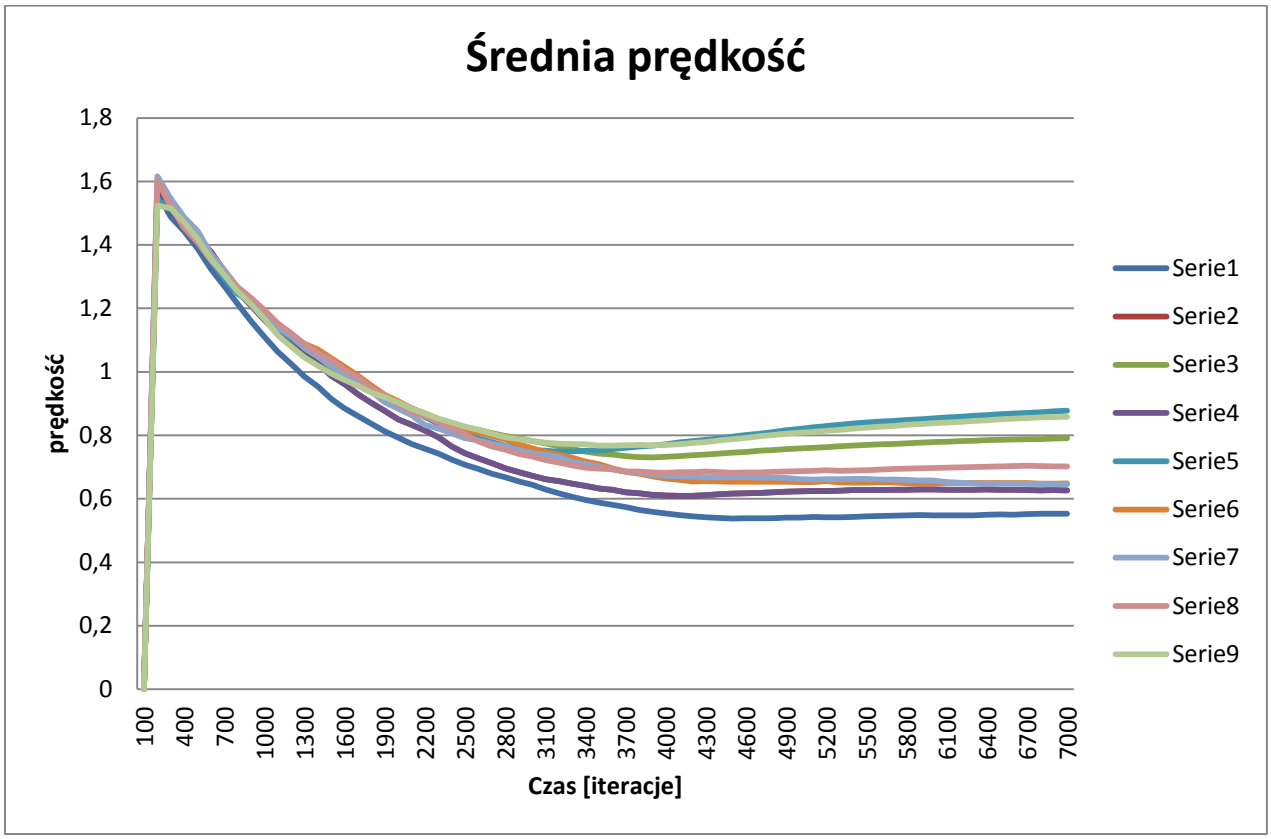
Poniższy wykres prezentuje zestawienie średnich czasów przejazdów pojazdów, mierzony po każdych 100 iteracjach.



Wykres 3 - Średni czas przejazdów dla symulacji na mapie Krakowa

5.2.2.3. *Średnie prędkości przejazdów*

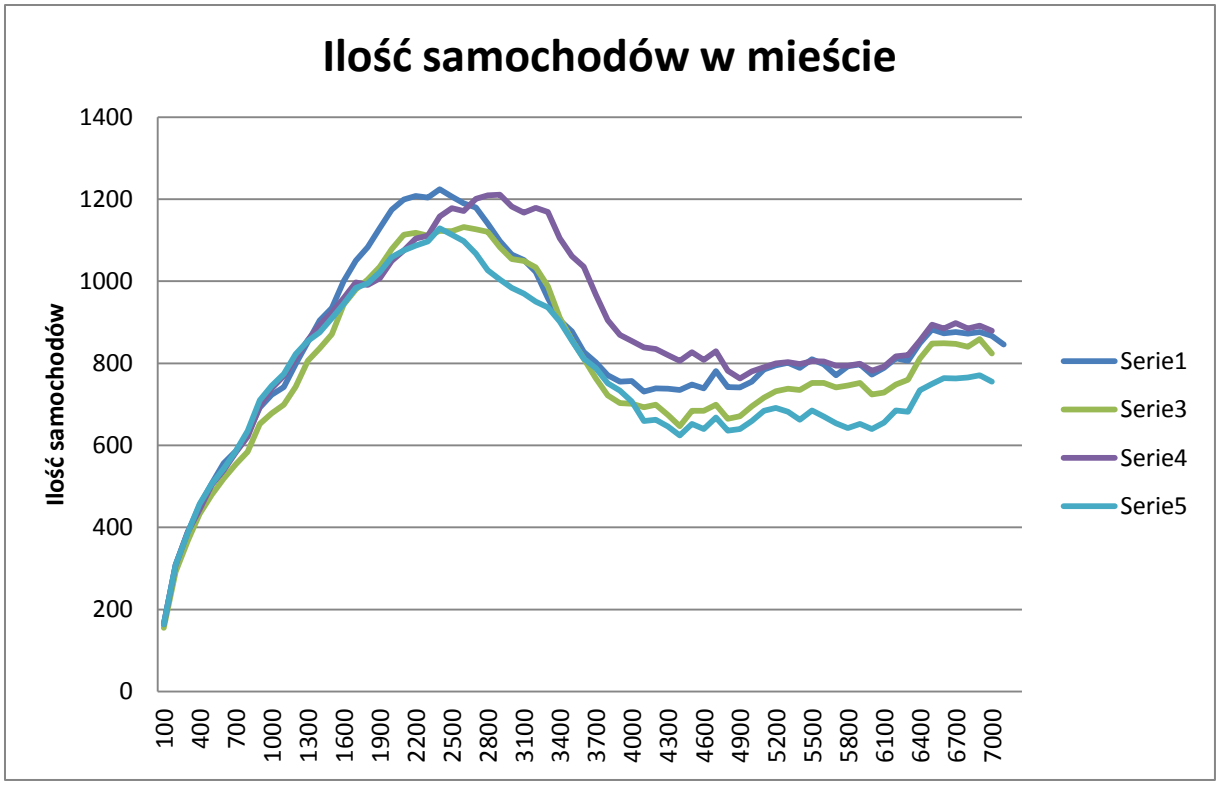
Poniższy wykres prezentuje zestawienie średnich prędkości pojazdów, mierzony po każdych 100 iteracjach.



Wykres 4 - Średnie prędkości pojazdów dla symulacji na mapie Krakowa

5.2.2.4. Ilość samochodów w mieście

- Serie 1 – Dane zebrane przy zastosowaniu starego algorytmu z algorytmem SOTL
- Serie 3 - Dane zebrane przy zastosowaniu algorytmu podstawowego, przy podziale na 4 klastry oraz przy użyciu algorytmu SOTL
- Serie 4 - Dane zebrane przy zastosowaniu algorytmu rozszerzonego, przy podziale na 7 klastrów oraz przy użyciu algorytmu SOTL
- Serie 5 - Dane zebrane przy zastosowaniu algorytmu rozszezrzonego, przy podziale na 6 klastrów oraz przy użyciu algorytmu SOTL



Wykres 5- Łączna ilość samochodów znajdujących się w danej iteracji w mieście.

5.2.3. Wnioski

Analizując zebrane dane, przedstawione na powyższych wykresach można wyciągnąć następujące wnioski:

- Przy użyciu algorytmu SOTL za stary algorytm optymalizacji zachowuje się nieco lepiej niż obie, zaimplementowane przez nas wersje algorytmu. Jednakże wyniki otrzymane przy użyciu naszych algorytmów nie odstają zbytnio od tych otrzymanych przy użyciu starych algorytmów. Zarówno jeśli chodzi o średnie prędkości jak i czasy przejazdu, wykresy otrzymane na podstawie danych otrzymanych podczas symulacji z użyciem naszych algorytmów są zbliżone kształtem, co może oznaczać iż algorytmu działają w sposób prawidłowy i optymalizują ruch, zgodnie ze swoim przeznaczeniem.
- Rozszerzona wersja naszego algorytmu wypada znacznie lepiej niż wersja podstawowa. Można na tej podstawie wysnuć wniosek iż dalszy rozwój naszego algorytmu, polegający na rozbudowaniu sposobu propagacji informacji w głąb klastra mógł by doprowadzić do zwiększenia efektywności działania algorytmu.
- Zmienną ilość klastrów testowaliśmy jedynie na mapie Krakowa, jako że przy mapie siatki nie było sensu stosować większej ilości klastrów ze względu na jej strukturę siatkową. Na podstawie wyników można zauważyć iż w przypadku rozszerzonego algorytmu ilość klastrów nie ma zbyt dużego wpływu na średnie czasy przejazdów. Jednakże ilość klastrów wydaje się być istotnym czynnikiem wpływającym na średnie prędkości pojazdów. Optymalna ilość klastrów jest zależna od mapy, na której przeprowadzana jest symulacja.
- Zauważyliśmy iż stosując podział na 6 klastrów otrzymujemy najlepsze wyniki. Zarówno czas przejazdu jak i średnia prędkość jest porównywalna dla algorytmu rozszerzonego z podziałem na 6 klastrów jak i dla starego algorytmu. W niektórych momentach nasz algorytm przynosi nawet lepsze rezultaty.

- Przy zastosowaniu algorytmu rozszerzonego (szczególnie dla większej ilości klastrów) możemy zaobserwować znacznie mniejszą ilość samochodów w całym mieście niż przy zastosowaniu starego algorytmu. Szczególnie dobre wyniki w tym kryterium osiągnęliśmy przy podziale na 6 klastrów.
- W przypadku algorytmu RL wyniki otrzymane przy użyciu naszych algorytmów nie wiele różniły się od tych, otrzymanych podczas symulacji wykorzystujących stare algorytmy. W niektórych przypadkach można nawet zauważyć że nasz algorytm rozszerzony przynosi lepsze rezultaty. Powyższa obserwacja również potwierdza iż zaimplementowany przez nas algorytm działa w sposób prawidłowy oraz że obrana przez nas metoda synchronizacji ruchu oraz optymalizacji świateł przynosi oczekiwane rezultaty.

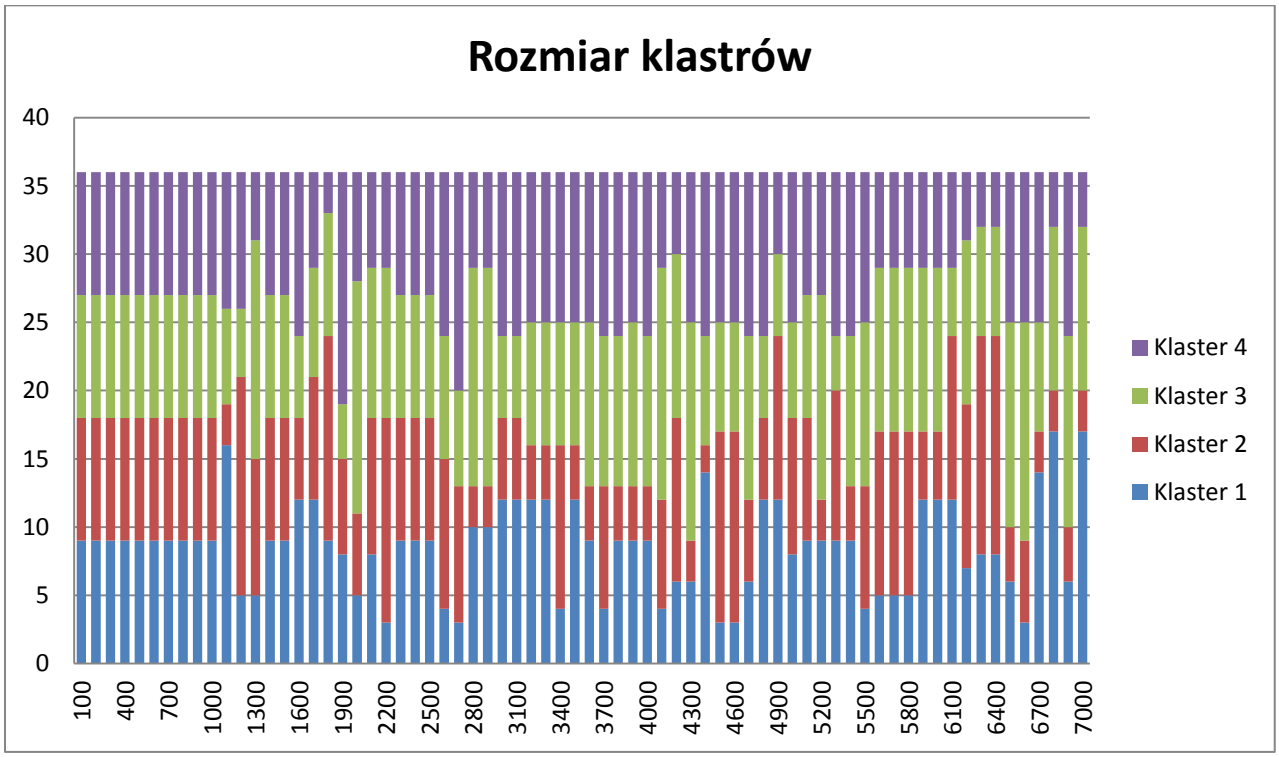
5.3. Testy – klastrowanie

Poniższe testy prezentują dane dotyczące klastrowania oraz hierarchizacji skrzyżowań (skrzyżowania główne). Przedstawiają rozmiary klastrów w kolejnych iteracjach oraz ilość wyborów danego skrzyżowania na główne skrzyżowanie klastra. Wykresy przedstawiają opisane wyniki dla konkretnej konfiguracji symulacji. Ze względu na dużą ilość danych zaprezentujemy poniżej jedynie niektóre z nich. Kompletne dane oraz wykresy znajdują się w załączonym arkuszu kalkulacyjnym.

5.3.1. Rozmiary klastrów

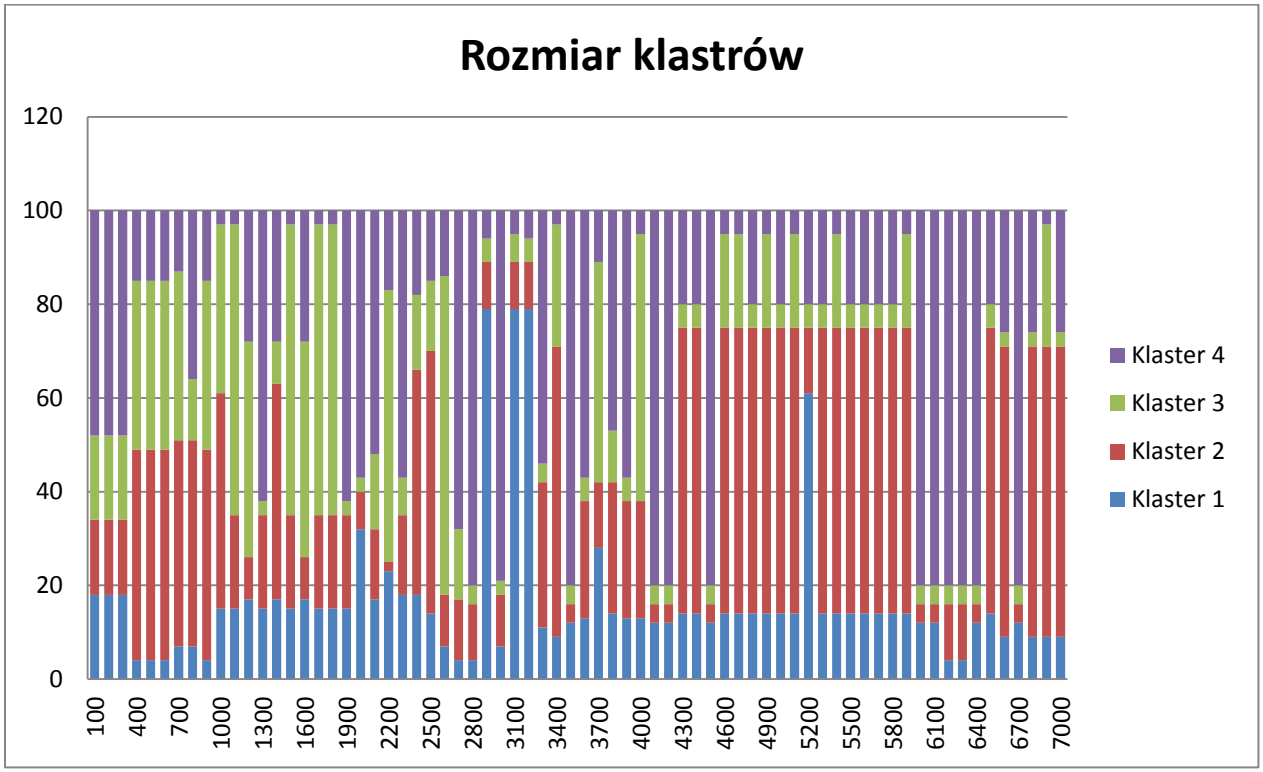
Poniższe diagramy przedstawiają rozmiary wszystkich klastrów w kolejnych iteracjach. Po każdych 100 iteracjach klastry przeliczane są na nowo, na podstawie wartości obliczonych miar centralności.

5.3.1.1. Mapa – siatka, algorytm podstawowy, 4 klastry, algorytm SOTL



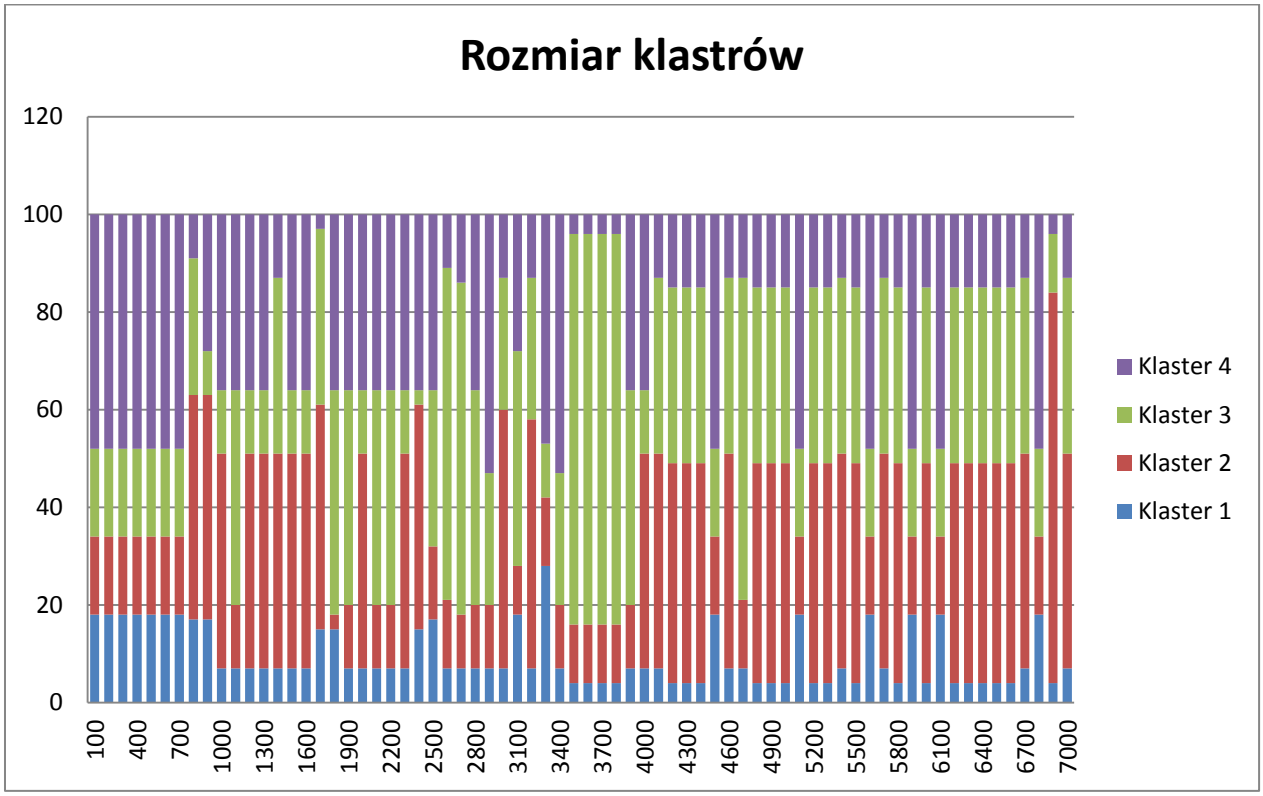
Wykres 6 - Wykres przedstawiający rozkład węzłów w klastrach

5.3.1.2. Mapa – Kraków, algorytm podstawowy, 4 klastry, algorytm SOTL



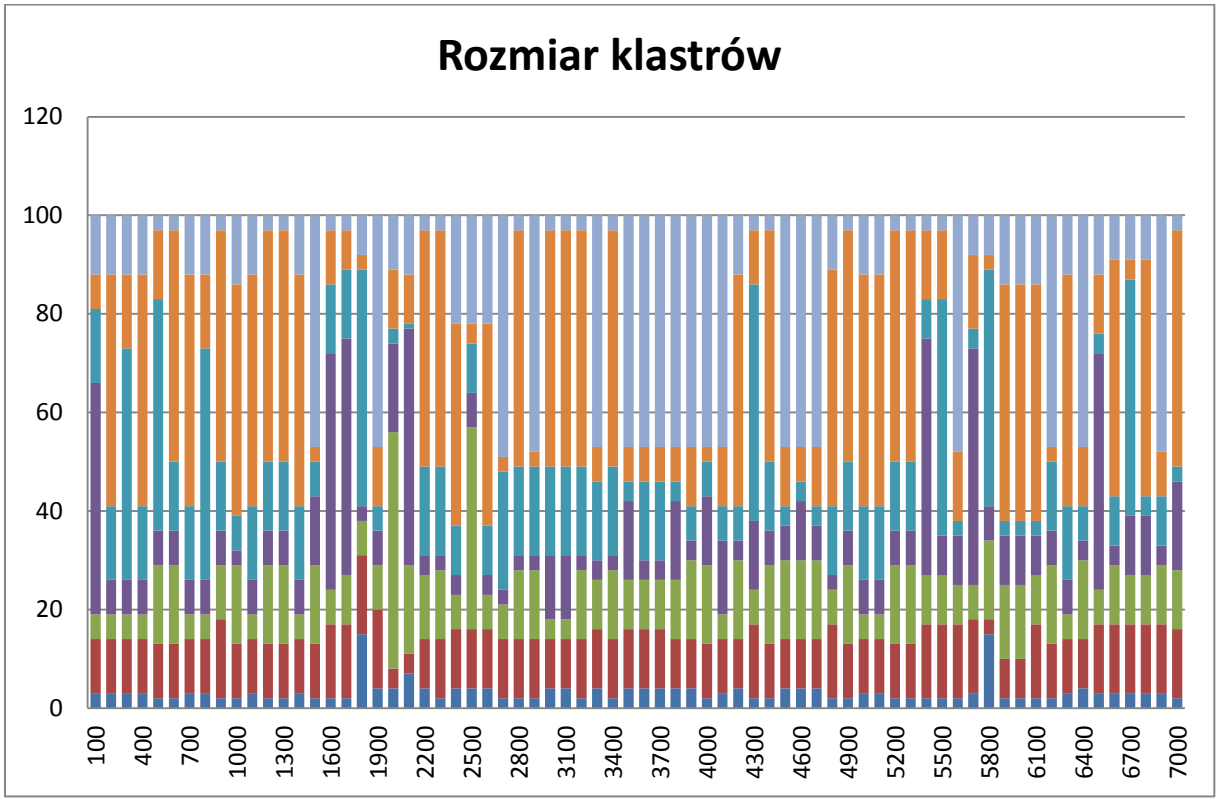
Wykres 7 - Wykres przedstawiający rozkład węzłów w klastrach

5.3.1.3. Mapa – Kraków, algorytm rozszerzony, 4 klastry, algorytm SOTL



Wykres 8 - Wykres przedstawiający rozkład węzłów w klastrach

5.3.1.4. Mapa – Kraków, algorytm rozszerzony, 7 klastrów, algorytm SOTL

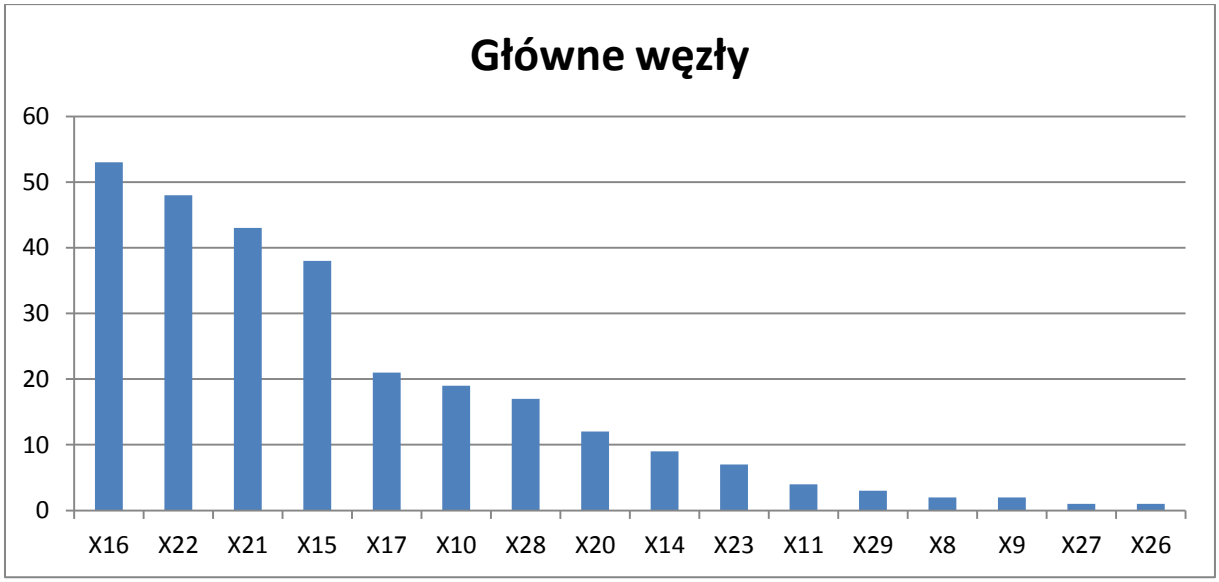


Wykres 9 - Wykres przedstawiający rozkład węzłów w klastrach

5.3.2. Hierarchizacja klastrów – przynależność do głównych węzłów

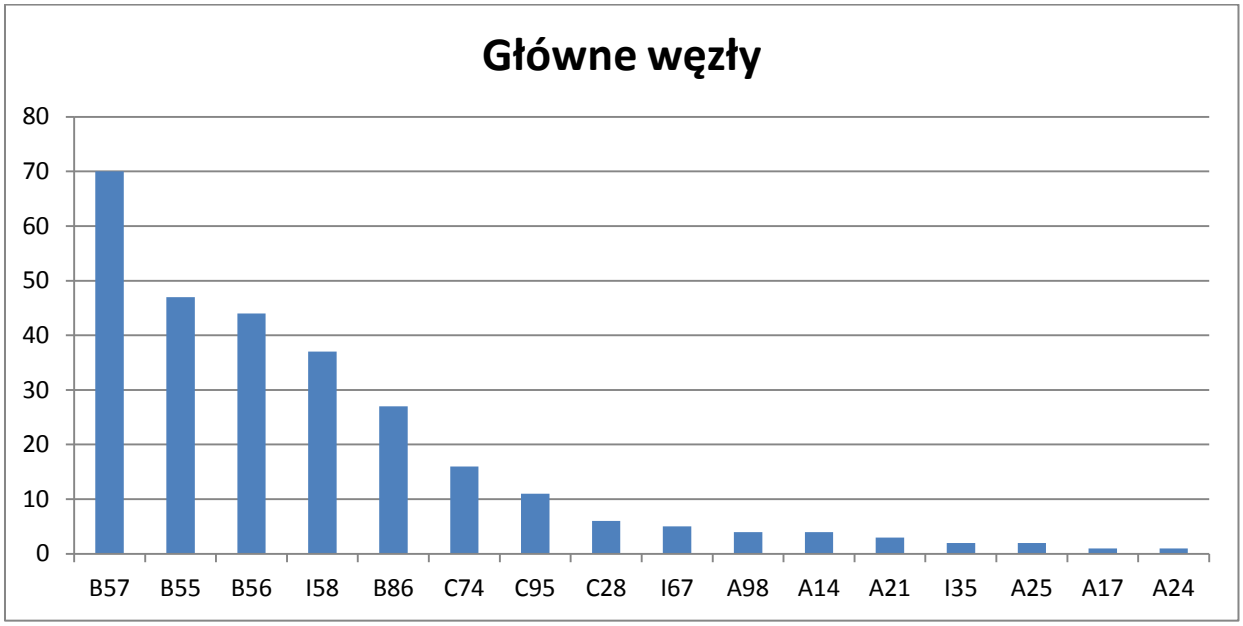
Poniższe diagramy przedstawiają ile razy dany węzeł został wybrany na główny węzeł w klastrze, a co za tym idzie miał najwyższą wartość wyliczonej miary.

5.3.2.1. Mapa – siatka, algorytm podstawowy, 4 klastry, algorytm SOTL



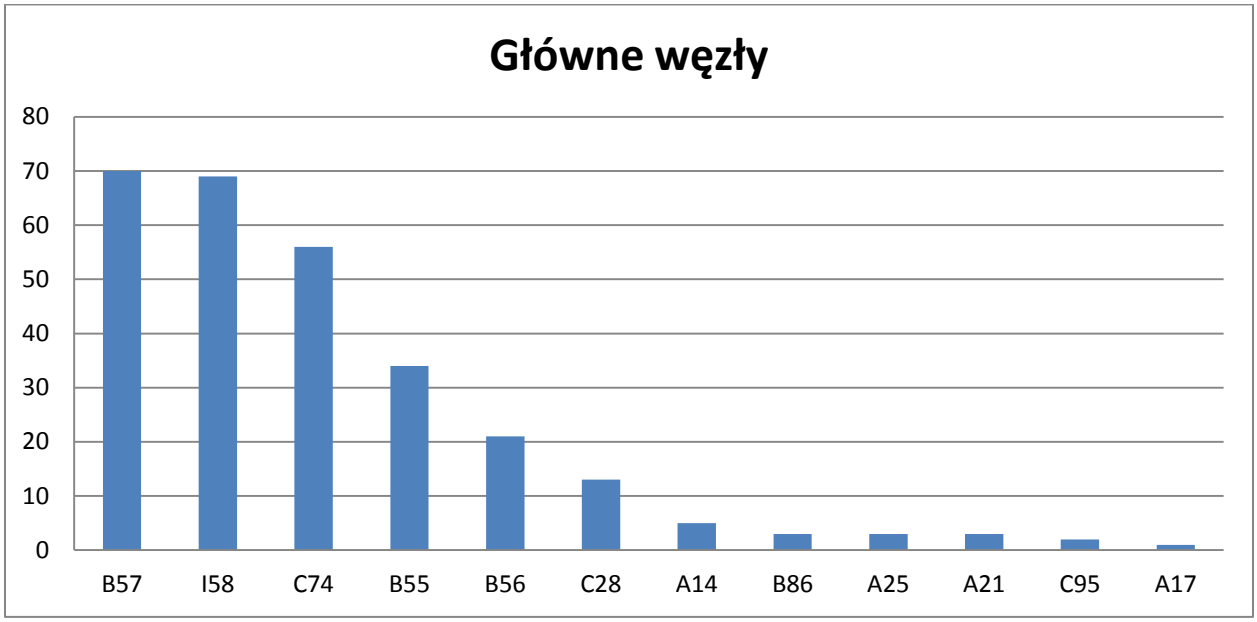
Wykres 10 - Ilość wystąpień węzła jako główny węzeł klastra.

5.3.2.2. Mapa – Kraków, algorytm podstawowy, 4 klastry, algorytm SOTL



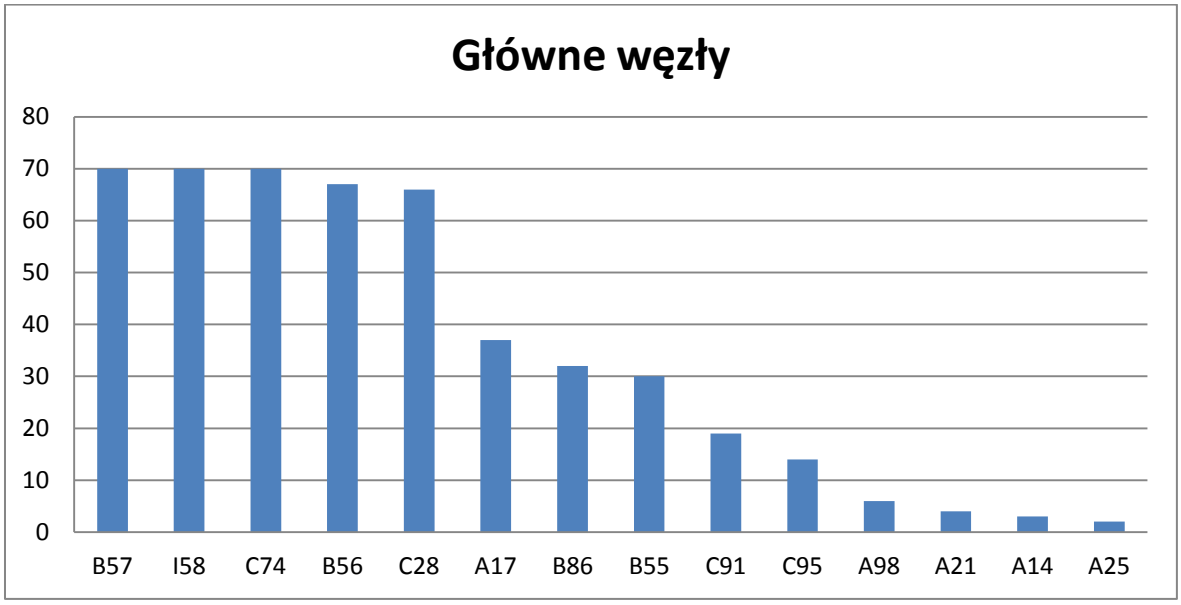
Wykres 11 - Ilość wystąpień węzła jako główny węzeł klastra.

5.3.2.3. Mapa – Kraków, algorytm rozszerzony, 4 klastry, algorytm SOTL



Wykres 12 - Ilość wystąpień węzła jako główny węzeł klastra.

5.3.2.4. Mapa – Kraków, algorytm rozszerzony, 7 klastrów, algorytm SOTL



Wykres 13 - Ilość wystąpień węzła jako główny węzeł klastra.

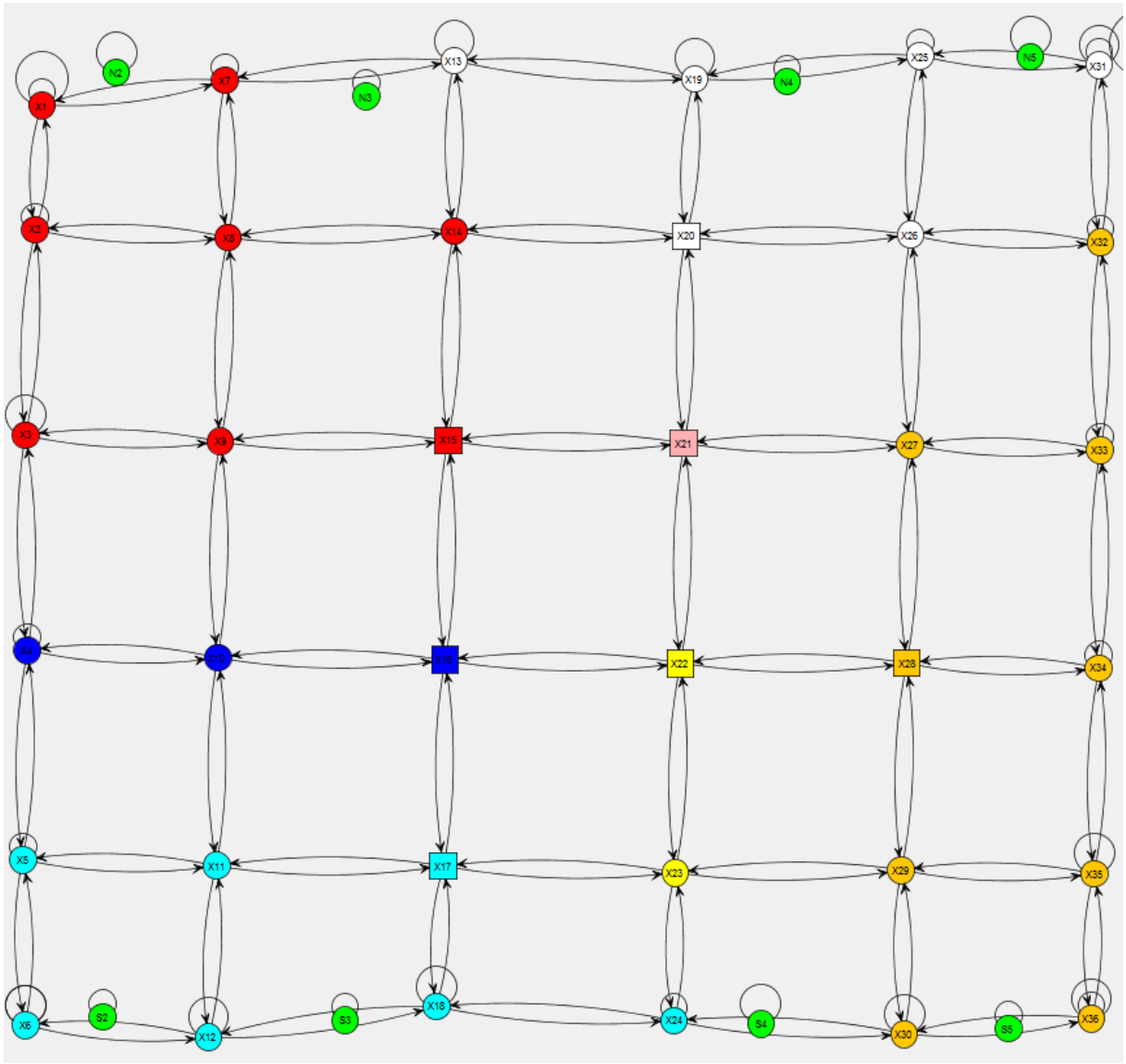
5.3.3. Wnioski

Analizując zebrane dane oraz przebieg przeprowadzanych symulacji można wyprowadzić następujące wnioski:

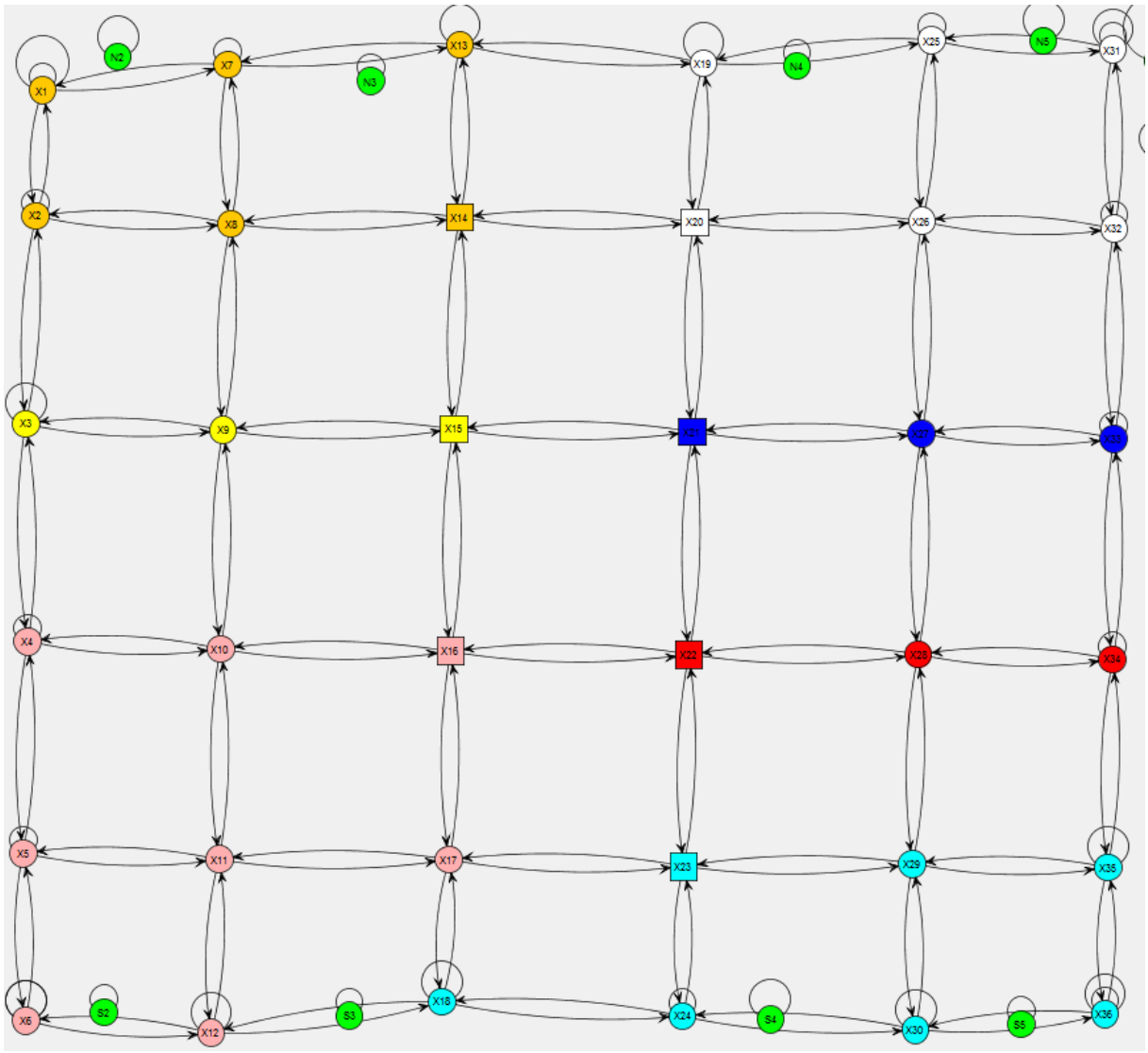
- Przy zastosowaniu algorytmu podstawowego ilość węzłów wybieranych na węzły główne klastrów jest większa. Oznacza to że podstawowa wersja algorytmu cechuje się większą różnorodnością oraz zmiennością ruchu.
- Powyższy wniosek wraz z obserwacjami symulacji pozwala stwierdzić iż podstawowa wersja algorytmu zachowuje się zgodnie z naszymi założeniami. Rozładowanie ruchu w przypadku algorytmu podstawowego polega na przeniesieniu części zakorkowania skrzyżowania na skrzyżowania sąsiednie, z tej też przyczyny podczas następnego przeliczania miar centralności skrzyżowania sąsiednie otrzymują wyższe miary, a co za tym idzie zostają wybierane na środki klastrów.
- Przy zastosowaniu algorytmu rozszerzonego mimo iż obserwujemy mniejszą różnorodność przy wyborze węzłów głównych możemy zauważyć lepszą poprawę ruchu w stosunku do algorytmu uproszczonego. Dzieje się tak dlatego że nie tylko skrzyżowania główne starają się zmniejszyć zakorkowanie swoich ulic ale także mają znaczny wpływ na zmianę konfiguracji ruchu, a co za tym idzie zatłoczenie ulic, skrzyżowań sąsiadujących.
- Można również zaobserwować iż w przypadku algorytmu rozszerzonego nie występuje tak duża różnorodność rozmiarów klastrów. Dzieje się tak dlatego iż główne węzły klastrów zmieniają się rzadziej utrzymując stały rozmiar klastrów przez dłuższy okres czasu.

6. Graf – podział grafu na klastry oraz hierarchizacja

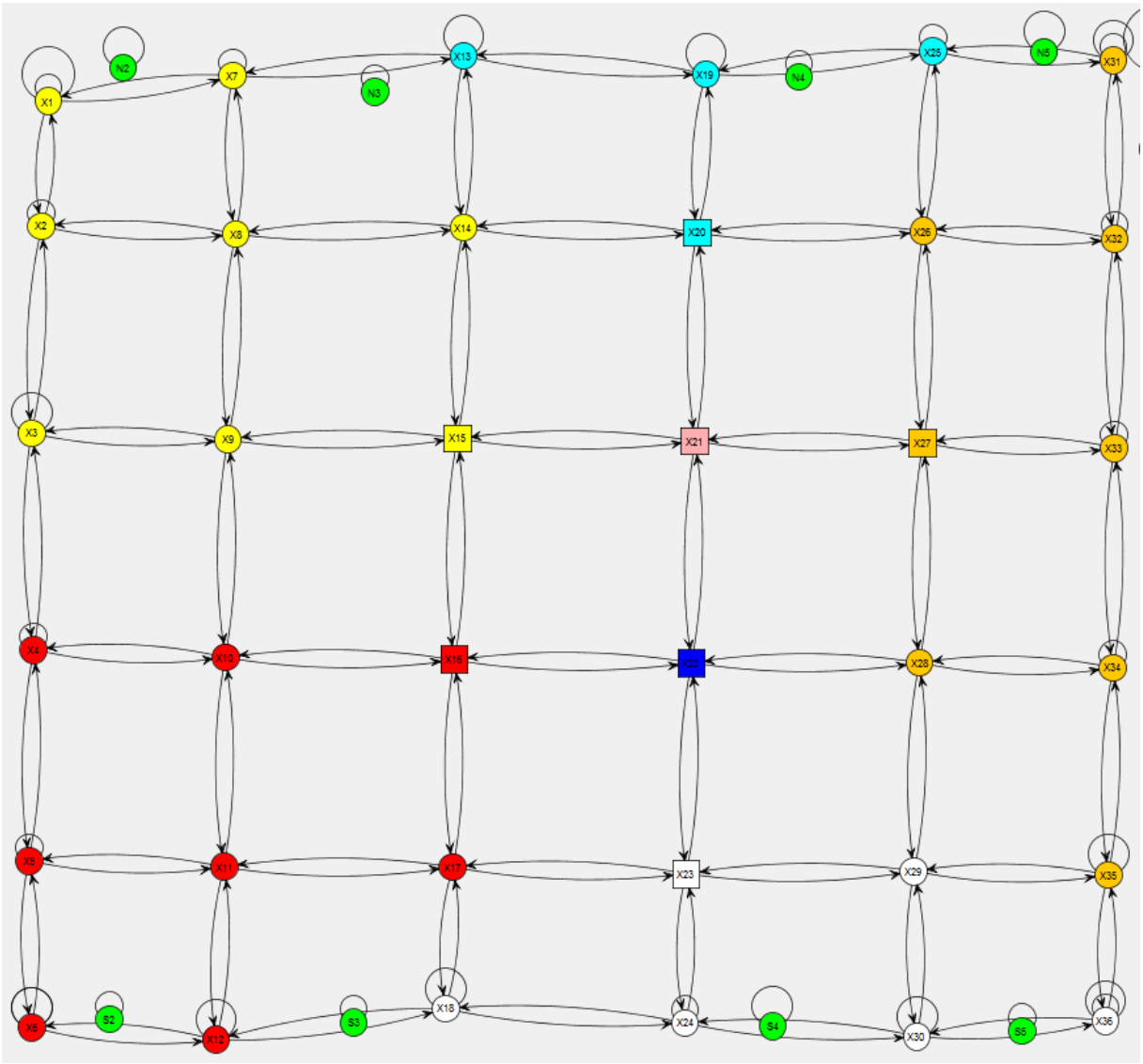
Poniższe rysunki przedstawiają zmianę podziału grafu na klastry na podstawie obliczanych miar w kolejnych iteracjach. Można zaobserwować jak zmieniają się rozmiary klastrów oraz jak przemieszczają się węzły główne klastrów. Węzły o kształcie kwadratowym reprezentują węzły główne klastrów. Pozostałe węzły to węzły podrzędne.



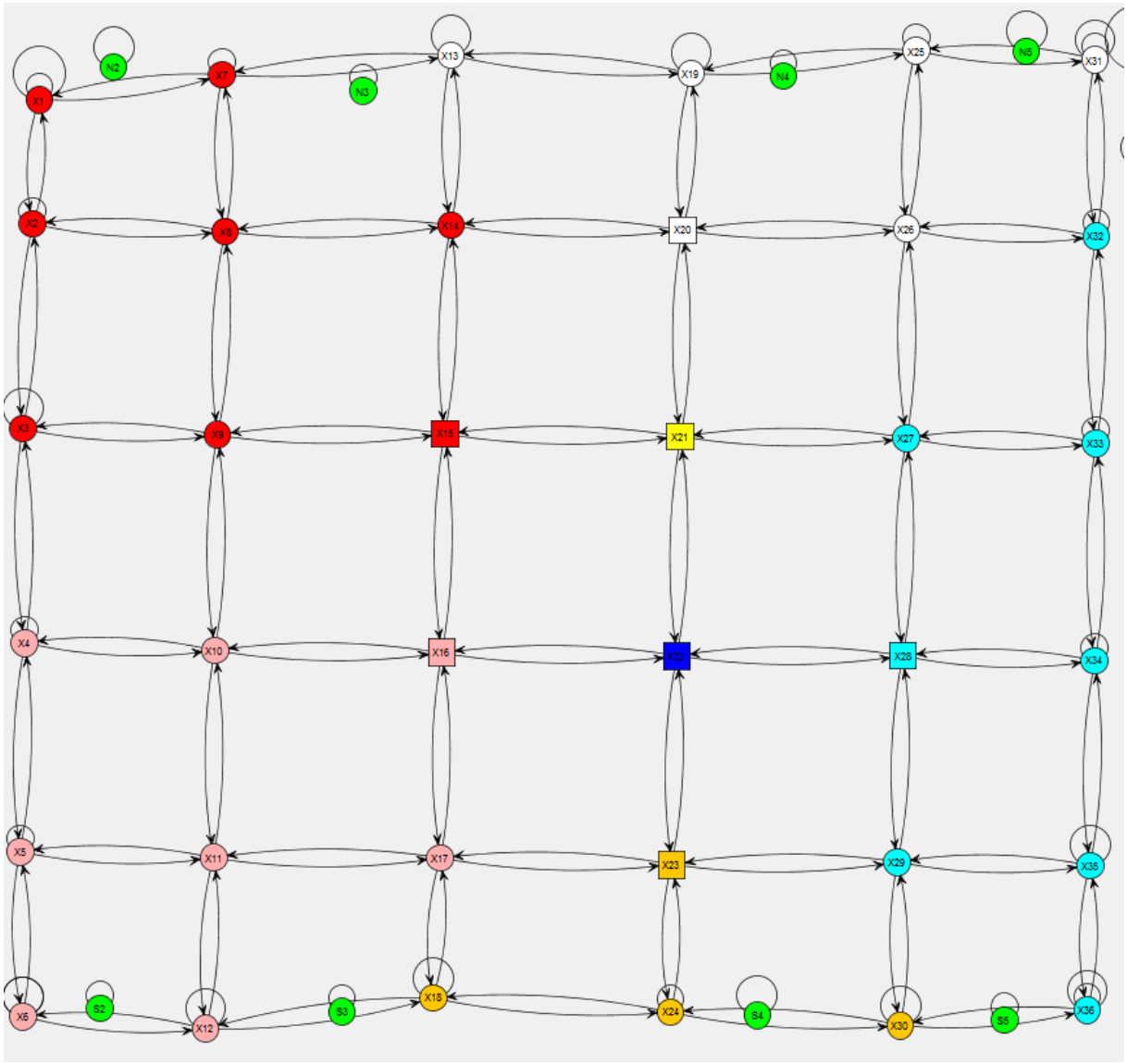
Rysunek 7 - Podział grafu na klastry przed rozpoczęciem symulacji (brak ruchu).



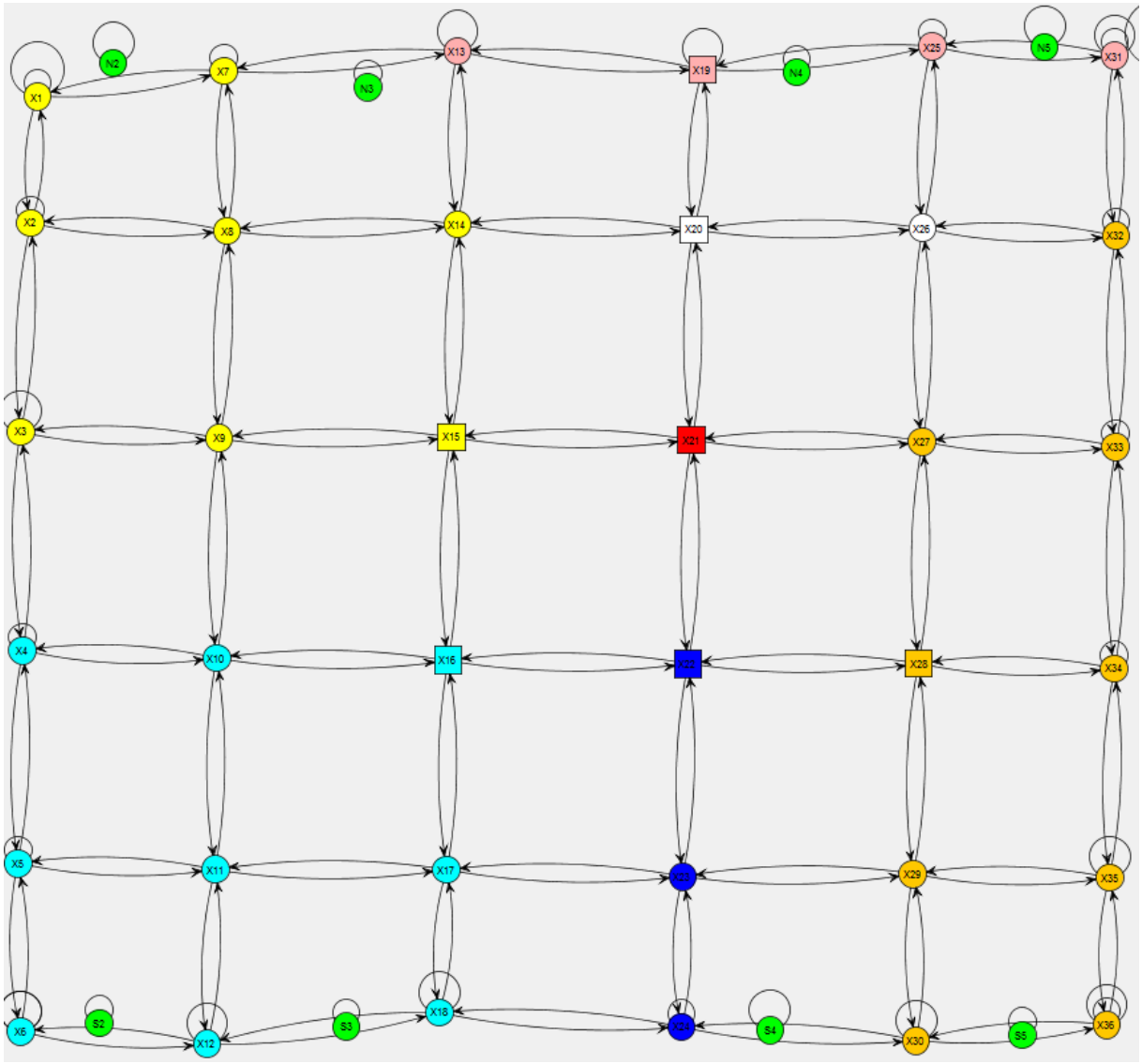
Rysunek 8 - Podział grafu na klastry po upływie 1500 iteracji.



Rysunek 9 - Podział grafu na klastry po upływie 3000 iteracji.



Rysunek 10 - Podział grafu na klastry po upływie 6000 iteracji.

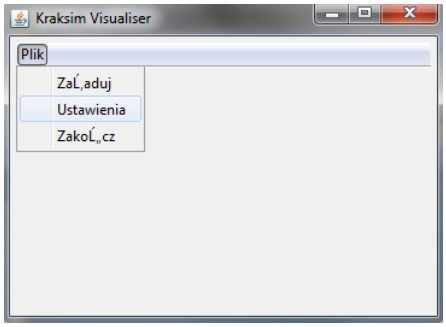


Rysunek 11 - Podział grafu na klastry po upływie 9000 iteracji.

7. Podręcznik użytkownika – uruchomienie oraz konfiguracje

7.1. Uruchomienie systemu

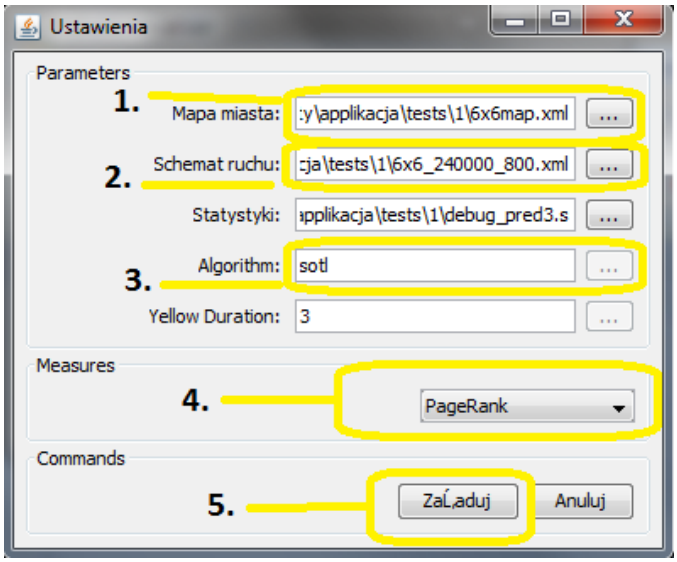
System możemy uruchomić poprzez uruchomienie pliku *kraksim.jar* lub też z poziomu eclips'a. Uruchomiony zostaje program Kraksim Visualiser a na ekranie widzimy małe okno.



Rysunek 12 - Menu głównego okna programu Kraksim

7.2. Konfiguracja oraz uruchomienie symulacji

Aby otworzyć okno ustawień konfiguracyjnych otwieramy menu i klikamy opcję Ustawienia (zaznaczona na Rysunku 12). Pojawia się okno z konfiguracjami. Możliwe iż okno będzie wymagało rozciągnięcia, tak aby możliwe było wyświetlenie wszystkich konrolek.



Rysunek 13 - Okno ustawień konfiguracji symulacji

1. Podajemy ścieżkę do pliku XML zawierającego mapę, której chcemy użyć w uruchamianej symulacji.
2. Podajemy ścieżkę do pliku XML zawierającego dane dotyczące ruchu.
3. Wybieramy rodzaj algorytmu (SOTL / RL).
4. Wybieramy miarę, która ma zostać zastosowana do obliczeń w symulacji.
5. Kliknięcie przycisku *Załaduj* uruchamia aplikację.

7.3. Plik konfiguracyjny

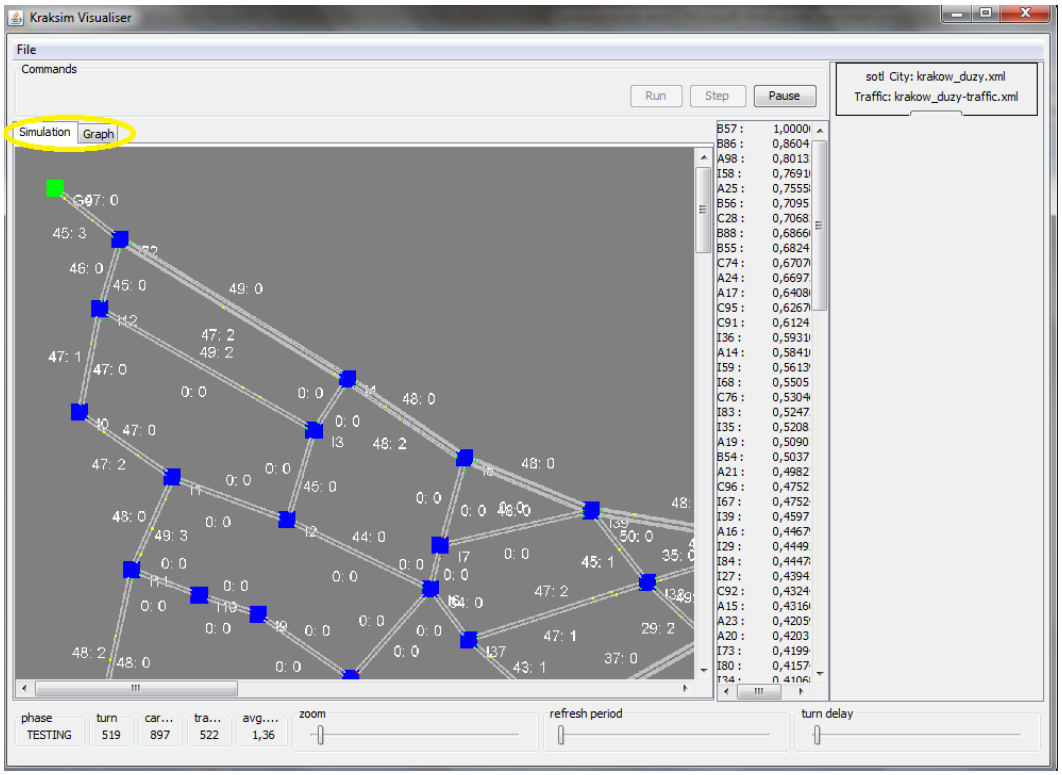
Możliwe jest również uruchomienie aplikacji po wcześniejszym sprecyzowaniu powyższych konfiguracji w pliku konfiguracyjnym. Plik konfiguracyjny – kraksim.properties powinien znajdować się w katalogu razem z aplikacją. Poza ustawieniami z poprzedniej wersji systemu rozszerzyliśmy go o ustawienia dotyczące optymalizacji ruchu za pomocą miar SNA. Ustawienia te są nastepujące:

- snaEnabled = true/false – ustawienie odpowiedzialne za to czy do optymalizacji ruchu wykorzystany będzie algorytm oparty o miary SNA.
- snaClusters = #clusters – ustawienie opisujące na ile klastrów ma zostać dzielony graf wykorzystywany do optymalizacji. Podajemy liczbę klastrow (np. 4).
- snaRefreshInterval = #interval – odpowiada za to jak często będzie dokonywane przeliczanie miar SNA, podział na klastry oraz konfigurowanie faz świateł na ich podstawie. Podajemy liczbę tur (np. 100).

Jeśli chcemy uruchomić symulacje korzystając z pliku konfiguracyjnego pomijając okno ustawień, klikamy opcję załaduj z menu przedstawionego na rysunku 12.

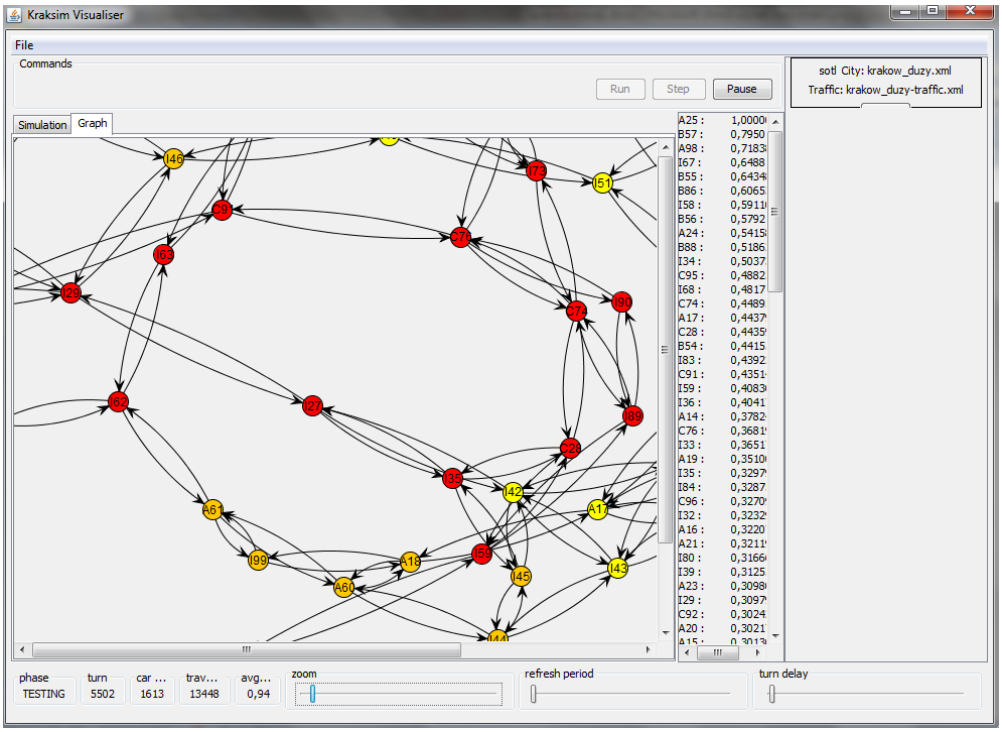
7.4. Symulacja

Gdy symulacja zostanie skonfigurowana oraz uruchomiona w głównym oknie aplikacji pojawia się mapa (zgodna z tą ustawioną przez nas podczas konfiguracji) oraz rozpoczyna się symulacja ruchu. Poruszające się pojazdy oraz natężenie ruchu na poszczególnych ulicach nanoszone są na mapę. Po prawej stronie mapy znajduje się lista zawierająca nazwy wierzchołków oraz wartości ich miar. Lista ta jest uaktualniana co 100 iteracji. Poniżej mapy znajdują się suwaki służące do zarządzania mapą oraz przebiegiem symulacji. Powyżej mapy (na Rysunku 8 zaznaczone żółtym okręgiem) znajdują się zakładki pozwalające na przełączanie się między widokiem mapy a widokiem grafu obrazującego wyliczone miary skrzyżowań (wierzchołków).



Rysunek 14 - Główne okno z widokiem mapy

Po naciśnięciu zakładki *Graph* mapę zastępuje graf. Użytkownik ma możliwość powiększania oraz pomniejszania grafu za pomocą scrolla. Możliwe jest również obracanie oraz rozciąganie grafu używając lewego przycisku myszy wciskając jednocześnie klawisz *Ctrl*.



Rysunek 15 - Główne okno z widokiem grafu

7.5. Uruchomienie z poziomu eclipsa.

Aby uruchomić symulację z poziomu eclipsa trzeba najpierw zaimportować projekt. Następnie, uruchamiamy projekt jako aplikację JAVA. Do uruchomienia aplikacji wybieramy klasę KraksimRunner. Po uruchomieniu otwiera się aplikacja i widzimy okno przedstawione w punkcie 7.1.

8. Implementacja

W celu zaimplementowania mechanizmu optymalizacji, opartego o miary SNA konieczne było rozszerzenie dotychczasowego projektu. W tym celu dodaliśmy klasy w pakiecie `pl.edu.agh.cs.kraksim.sna.centrality`, z których najważniejsze opisane są poniżej:

CentrallityCalculator

Klasa odpowiedzialna za przeliczanie miar centralności. Dostarcza funkcjonalności niezbędnych do przekształcenia struktury **City**, zawierającej informacje dotyczące miasta, na strukturę **Graph**, z biblioteki JUNG. Dzięki takiej transformacji możliwe jest wyświetlanie odpowiedniej struktury grafu oraz obliczenie miar centralności dla poszczególnych węzłów (skrzyżowań). Zawiera również metody pomocnicze, wykorzystywane przy konstrukcji grafu takie jak np. normalizacja miar.

KmeansClustering

Jest to klasa, która odpowiedzialna jest za podział skonstruowanego przez nas grafu na odpowiednią (zadaną w konfiguracji) liczbę klastrow. Dostarcza również mechanizmów wykorzystywanych przez agentów (skrzyżowania) podczas ustalania konfiguracji świetlnej, związanych z klastrami, takich jak sprawdzanie przynależności skrzyżowania do klastra bądź wskazywanie węzła nadrzędnego (środka klastra) dla danego skrzyżowania.

CentrallityStatistics

Klasa pomocnicza, wykorzystywana do zbierania interesujących nas informacji dotyczących przebiegu działania algorytmu optymalizacji ruchu. Zebrane wyniki co określony czas zapisywane są do plików.

OptimalizationInfo

Klasa odpowiedzialna za przechowywanie informacji dotyczących zmian, które dane skrzyżowanie chcę wprowadzić w swojej konfiguracji świetlnej. Informacje te są przesyłane pomiędzy agentami (skrzyżowaniami).

Aby możliwe było działanie naszego algorytmu musieliśmy rozszerzyć również klasę skrzyżowania (**Intersection**) o metody odpowiedzialne za ustalanie konfiguracji świateł oraz o propagację informacji do pozostałych skrzyżowań.

Aby możliwa była wizualizacja grafu konieczna była również ingerencja w klasy odpowiedzialne za przebieg symulacji oraz za wizualizację głównego panelu. Dodana zostala oddzielna zakładka, wizualizujaca graf oraz panel przedstawiający aktualne wartości miar dla wszystkich skrzyżowań w mieście. Zaimplementowane zostały również mechanizmy odpowiedzialne za odświeżanie grafu.

9. Bibliografia

1. Java Universal Network/Graph Framework - http://jung.sourceforge.net/
2. SNA - http://en.wikipedia.org/wiki/Social_network_analysis
3. Betweenness Centrallity - http://en.wikipedia.org/wiki/Betweenness_centrality
4. Page Rank - http://en.wikipedia.org/wiki/PageRank
5. Modelowanie I optymalizacja ruchu miejskiego przy użyciu wybranych technik – praca magisterska Bartosza Rybackiego
6. Modyfikacje systemu Kraksim – dokumentacja Łukasza Dziewońskiego i Macieja Zalewskiego
7. Referat dotyczący podziału grafu (Graph partitioning) dostarczony w poprzednim semestrze.