Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa magisterska

na kierunku Informatyka
w specjalności Inteligentne systemy

Uczenie ze wzmocnieniem sterowników sygnalizacji świetlnej

Szymon Kubiszewski
Numer albumu 300242

promotor
dr hab. inż. Mariusz Kamola

WARSZAWA 2024

Uczenie ze wzmocnieniem sterowników sygnalizacji świetlnej

Streszczenie.

Tematem pracy magisterskiej było zbadanie możliwości stworzenia uniwersalnego rozwiązania pozwalającego na kontrolowanie sygnalizacji świetlnych w celu minimalizacji czasu traconego na oczekiwanie na przejazd. System komunikujących się ze sobą sterowników miał umożliwiać wykorzystanie raz stworzonego mechanizmu na skrzyżowaniach o różnych układach. W celu stworzenia takiego rozwiązania, postanowiono wykorzystać algorytm wieloagentowego głębokiego uczenia ze wzmocnieniem.

W pracy zawarto wyniki przeprowadzonej analizy źródeł poruszających tematykę dziedziny sterowania ruchem za pomocą sygnalizacji świetlnej na skrzyżowaniach oraz wykorzystywania wybranych algorytmów w tym celu. Na podstawie uzyskanych informacji określono główne założenia projektu wraz z oczekiwanymi wynikami. Pokrótce opisano także wykorzystany symulator ruchu drogowego SUMO. Aby usystematyzować zdobytą wiedzę i uchwycić lepszy obraz zagadnienia stworzono prototyp rozwiązania, którego obserwacja pozwoliła na ostateczne zaplanowanie projektu. Opisano ważniejsze szczegóły implementacyjne oraz użyte mechanizmy, a także skomponowany program uczenia. Przedstawiono również kilkanaście stworzonych siatek ulic.

Jako wynik przeprowadzonego za pomocą algorytmu *Double deep Q-learning* procesu uczenia uzyskano sześć modeli, które następnie porównano w serii testów ze standardowym akomodacyjnym kontrolerem sygnalizacji świetlnej. Za ich pomocą zebrano obszerne dane dotyczące przejazdów wykonanych przez każdy z pojazdów. Zgromadzono także informacje dotyczące kolejek na drogach dojazdowych do skrzyżowań.

Na podstawie otrzymanych danych przeprowadzono analizę prowadzącą do zweryfikowania zaprojektowanego rozwiązania.

Słowa kluczowe: Wieloagentowe głębokie uczenie ze wzmocnieniem, double DQN, inteligentne systemy transportowe, sygnalizacja świetlna, optymalizacja przepływu ruchu

Reinforcement learning of traffic light controllers

Abstract.

The main purpose of this thesis was to investigate the possibility of creating a universal solution which allows to control traffic lights in order to minimize the waiting for the green light. The system of communicating controllers was intended to enable the use of the previously created mechanism at intersections with different layouts. In order to create such a solution, it was decided to use a multi-agent deep reinforcement learning algorithms.

The work contains the results of the performed analysis of sources dealing with traffic control using traffic ligths at intersections and the use of selected algorithms for this purpose. Based on the information obtained, the main assumptions of the project were determined along with the expected results. The SUMO traffic simulator used in the project was also briefly described. In order to systematize the acquired knowledge and capture a better picture of the issue, a prototype solution was created. Subsequent verification of its performance allowed for a more intrinsic perspective of the task. The work desribes the most important details of the implementation and the mechanisms used, as well as programme used for learning the neural network. Several street grids created for this purpose were also presented.

As a result of the learning process carried out using the Double deep Q-learning algorithm, six models were obtained. They were then compared in a series of tests to a a standard actuated traffic light controller. The research allowed for collection of extensive data on the journeys of individual vehicles. Information on queues of cars on access roads was also collected.

Based on the data obtained, an analysis was carried out to verify the designed solution.

Keywords: Multiagent deep reinforcement learning, double DQN, intelligent transportation systems, traffic lights, traffic flow optimization

Spis treści

1. Wstęp

1.1. Motywacja pracy

W dzisiejszym szybko urbanizującym się świecie liczba pojazdów na drogach stale rośnie, co nie zawsze idzie w parze z rozwojem infrastruktury drogowej. Przekłada się to na coraz większe zakorkowanie miast i zwiększenie średnich czasów przejazdów. Intuicyjna propozycja rozwiązywania tych problemów poprzez dodawanie kolejnych pasów ruchu nie zawsze jest możliwa do zastosowania i może prowadzić do dalszych komplikacji. Poszerzanie dróg w miastach najczęściej wiąże się z dużymi nakładami finansowymi i kosztami społecznymi, a czasem nie jest możliwe do zrealizowania z powodu ograniczonego miejsca. Jak pokazuje paradoks Braessa[1] oraz zjawisko ruchu wzbudzonego (ang. *induced demand*) [2], dodawanie kolejnych pasów ruchu może prowadzić do większych korków [3]. Pewnym jest, że o ile nie zawsze zachodzi taka zależność i często poszerzanie jezdni przynosi spodziewane efekty, tak obecne trendy urbanistyczne w miastach Unii Europejskiej jednoznacznie wskazują na chęć postępującego ograniczania liczby samochodów [4].

Duży ruch drogowy wiąże się z szeregiem konsekwencji mających znaczny wpływ na życie mieszkańców, jak i na środowisko. Emisja spalin, hałas, wypadki drogowe to tylko niektóre z bezpośrednich skutków szkodliwych dla zdrowia człowieka. Koszty obejmują także czas stracony w zatorach czy też pieniądze z powodu dodatkowego zużycia paliw. Ponadto istotne są utracone możliwości w sferach pracy i odpoczynku wynikające z długiego przebywania w ruchu ulicznym. Przykładowo zmotoryzowani mieszkańcy Warszawy mogą tracić ponad 8, a Wrocławia prawie 10 dni rocznie na stanie w korkach [5]. Wpływem nieefektywnego ruchu drogowego na środowisko są niepotrzebne emisje dwutlenku węgla oraz innych gazów cieplarnianych, a także marnowanie zasobów naturalnych, które są ograniczone i mogą w przyszłości ulec wyczerpaniu.

Rosnąca populacja jak i liczba pojazdów w miastach razem z obecnymi trendami urbanistycznymi prowadzącymi do ograniczenia transportu indywidualnego sprawia, że badanie potencjału nowych metod sterowania ruchem staje się szczególnie perspektywiczne. Usprawnienie ruchu przez minimalizację czasu oczekiwania na przejazd na skrzyżowaniach pomogłoby ograniczyć negatywny wpływ na środowisko, a także na mieszkańców miast. Dodatkowo pozwoliłoby na ograniczenie skutków wzrostu liczby pojazdów.

Wykorzystanie w tym celu sztucznej inteligencji może przynieść pozytywne wyniki. W ostatnich latach zaobserwować można szczególnie szybki rozwój tej dziedziny, potrafiącej przekształcać przeróżne aspekty codziennego życia. Najbardziej rozpoznawalnymi są bez wątpienia pojazdy autonomiczne, chatboty oparte o modele językowe czy też oprogramowanie służące do generowania grafiki. Postęp ten jest napędzany zarówno przez innowacje w uczeniu maszynowym, jak i przez szczególny wzrost mocy obliczeniowej. Zastosowanie uczenia ze wzmocnieniem w zarządzaniu ruchem może przyczynić się do lepszego zrozumienia zasad przepływu pojazdów, co może przełożyć się na bardziej efektywną urbanistykę pod kątem projektowania zasad transportu osobowego.

1.2. Cel pracy

Celem pracy dyplomowej było zaproponowanie rozwiązania wykorzystującego uczenie ze wzmocnieniem pozwalającego na sterowanie ruchem pojazdów na skrzyżowaniach drogowych z sygnalizacją świetlną. Korzystając z uczenia ze wzmocnieniem został stworzony agent kierujący światłami drogowymi, minimalizujący średni czas oczekiwania na przejazd dla pojazdów uczestniczących w ruchu. Kluczowym założeniem projektu było stworzenie takiego agenta, który po uprzednim wytrenowaniu, możliwy będzie do zastosowania na skrzyżowaniach o różnorodnych topologiach.

Podczas projektowania rozwiązania brano pod uwagę zarówno zdolność agenta do działania w różnorodnych sytuacjach, jak i jakość uzyskanego przez niego sterowania, która oceniana była za pomocą mierzonych metryk: średniego i maksymalnego opóźnienia pojazdu, maksymalnej liczby pojazdów oczekujących i innych. Ważnym aspektem była także szybkość działania.

Na podstawie przeprowadzonej analizy literaturowej, w odróżnieniu od innych prac, zdecydowano się na bardziej podejście wykonalne w prawdziwym świecie. Główną różnicą były dane, które uzyskiwał agent. Pochodziły one wyłącznie ze źródeł, które są dostępne dla rzeczywistych systemów sterowania ruchem. Nie mogły więc to być dokładne informacje na temat położeń, prędkości ani tras pojazdów, będące obecnie koncepcjami w ramach inteligentnych miast.

Ze względu na brak możliwości testowania rozwiązania w świecie rzeczywistym, konieczny był wybór odpowiedniego symulatora odwzorowującego w poprawny sposób ruch drogowy. Musiał on także dostarczać rzeczywiste dane pomiarowe dotyczące ruchu, odwzorowujące te możliwe do uzyskania w prawdziwym świecie.

1.3. Etapy pracy

Pierwszym etapem pracy w ramach niniejszej pracy magisterskiej była analiza stanu badań i techniki dotyczącej sterowania ruchem. Zapoznano się z obowiązującymi zasadami projektowania skrzyżowań oraz sposobami sterowania światłami sygnalizacji drogowej, zarówno w ramach pojedynczego skrzyżowania, jak i całych sieci. Zaznajomiono się z historią dziedziny sterowania ruchem. Przeanalizowano aktualne prace naukowe dotyczące zastosowania uczenia ze wzmocnieniem, także w kontekście sterowania ruchem drogowym. Poznano wprowadzone w nich rozwiązania, wykorzystane narzędzia oraz potencjalne kierunki rozwoju. Ustalono również typy skrzyżowań, które dominują w współczesnych miastach. Znaleziono wyniki pomiarów ruchu ulicznego w miejscowościach dostarczających wgląd w złożoność zadania. Zapoznano się ze sposobami pomiaru ruchu.

W kolejnym etapie przygotowano sztuczne siatki ulic oraz takie, które odwzorowują rzeczywiste układy ulic wykorzystywane w kolejnych etapach pracy w procesie uczenia oraz testowania tworzonego agenta. Połączono ze sobą komponenty projektu i stworzono prototyp mający za zadanie sterować pojedynczym skrzyżowaniem. Pozwoliło to na dokładne przetestowanie środowiska uczącego, potwierdzenie dobrego wyboru symulatora i

przeanalizowanie ograniczeń prototypu. Dzięki temu, w ramach tego etapu opracowano także główne założenia ostatecznego projektu.

Następna część nastawiona była na implementację założonego projektu. Korzystając z wieloagentowego uczenia ze wzmocnieniem stworzono rozwiązanie pozwalające na sterowanie światłami ruchu drogowego za pomocą homogenicznych agentów. Takie podejście pozwalało na zastosowanie grupę raz nauczonych agentów na skrzyżowaniach o różnych układach. Aby potwierdzić poprawne działanie założonego systemu kontroli ruchu przeprowadzono serię testów, których wyniki porównano ze światłami adaptacyjnymi.

2. Analiza stanu badań i techniki

W tym rozdziale przybliżono pokrótce historię dziedziny sterowania sygnalizatorami świetlnymi na skrzyżowaniach dróg. Przedstawiono także wyniki przeprowadzonego przeglądu literatury dotyczącego tego zagadnienia.

Ponieważ koszty wysokiego natężenia ruchu są zarówno łatwo mierzalne finansowo, jak i powszechnie odczuwalne przez populację, tematyka optymalizacji przepływu pojazdów w miastach jest szeroko badana. Na przestrzeni lat powstało wiele algorytmów sterujących pojedynczymi skrzyżowaniami, a także skomplikowanych systemów nadzorujących całe sieci dróg miejskich. W miarę rozwoju dziedziny, rozwiązania zmieniały swoją charakterystykę z ręcznie tworzonych scenariuszy uruchamianych w zależności od natężenia ruchu do sygnałów generowanych dynamicznie w oparciu o dane odczytywane w czasie rzeczywistym z czujników.

2.1. Rys historyczny

Pierwsza, sterowana ręcznie gazowa sygnalizacja świetlna została zainstalowana w Londynie w 1868 roku [6]. Wprowadzenie najpierw elektrycznych, a następnie automatycznych (czasowych) sygnalizacji świetlnych (lata dwudzieste) wiązało się z ogromnymi oszczędnościami, związanymi z brakiem konieczności obsadzenia każdego urządzenia przez przeszkolony personel [7]. Początkowo, wszystkie światła przełączały się w jednym momencie, jednak wprowadzenie tzw. „zielonej fali" w kolejnych latach pozwoliło na znaczne zwiększenie przepustowości ruchu w dużych miastach Stanów Zjednoczonych oraz zachodniej Europy, które już wtedy borykały się z problemem zatłoczonych dróg. Upowszechnienie komputerów w USA w latach 50-tych przyczyniło się do dalszego usprawnienia sterowania ruchem, wraz z powstaniem wyspecjalizowanych w tym celu sensorów. Wtedy też pojawiły się próby usystematyzowanego matematycznego opisu zagadnienia, wykorzystujące na przykład zasady mechaniki płynu do opisu poruszających się pojazdów [8]. Lata 70-te przyniosły rozwój pierwszych scentralizowanych systemów zarządzania ruchem. W kolejnych dziesięcioleciach, wraz z rozwojem wyspecjalizowanych detektorów, na przykład pętli indukcyjnych, radarów oraz kamer ruchu postępował także rozwój stosowanych systemów świetlnych. W dzisiejszych czasach, na obszarach z dużym ruchem drogowym stosuje się sygnalizatory akomodacyjne (odpowiednio wydłużające lub skracające czas zielonego światła w zależności od aktualnego natężenia ruchu), najczęściej skoordynowane między sobą w celu zapewnienia jak największej przepustowości ciągów komunikacyjnych [9].

2.2. Przegląd literatury

Pierwsze scentralizowane systemy adaptacyjnego sterowania światłami w sieci miejskiej, tworzone odpowiednio w Wielkiej Brytanii oraz Australii: SCOOT (*Split Cycle Offset Optimisation Technique*) [10] oraz SCATS (*Sydney Coordinated Adaptive Traffic System*) [11] zaczęły powstawać już we wczesnych latach 80-tych. Oba rozwiązania rozwijane są do teraz i stosowane w wielu miastach na całym świecie. W dużym uproszczeniu, zarówno SCOOT

jak i SCATS korzystają z różnych metryk natężenia ruchu, uzyskiwanych na podstawie danych pochodzących z czujników drogowych, jak i danych historycznych. Uzyskane charakterystyki są przetwarzane za pomocą modelu matematycznego, który następnie w zależności od wyniku modyfikuje aktualne sygnały świetlne [12].

Innym, względem klasycznych, podejściem jest algorytm *Max-Pressure* [13]. Za pomocą tej strategii każdemu pasowi ruchu przydziela się „ciśnienie" proporcjonalne do natężenia ruchu na danym pasie. Pierwszeństwo przejazdu przydzielane jest tym kierunkom, których sumaryczne ciśnienie ma największą wartość. W miarę opuszczania skrzyżowania przez przejeżdżające pojazdy, priorytet danego pasa jest zmniejszany. Takie podejście pozwala na skuteczne sterowanie ruchem. Algorytm ten szczególnie znacząco zwiększa przepustowość pasów o większym natężeniu ruchu. Dodatkową zaletą jest mniejsza względem innych systemów liczba wymaganych czujników do poprawnego działania oraz zdolność do reagowania na zmieniające się warunki ruchu.

W opracowaniach [14], [15] przedstawiono rozwiązania korzystające z mieszanego programowania liniowego całkowitoliczbowego (MILP). Mimo dużej złożoności obliczeniowej algorytmów rozwiązujących takie zadania, w pracy [16] udowodniono, że czasy znajdowania rozwiązań pozwalają na ich zastosowanie w systemach czasu rzeczywistego.

W ostatnich latach, wraz ze wzrostem wydajności komputerów i rozwojem symulatorów ruchu drogowego rośnie również liczba opracowań korzystających z różnych metod sztucznej inteligencji. Wśród przykładów wyróżnić można algorytmy ewolucyjne oraz wykorzystującę logikę rozmytą (ang. *fuzzy logic*) [17]. Inną, szczególnie badaną gałęzią jest zastosowanie uczenia ze wzmocnieniem, w szczególności w połączeniu z głębokimi oraz rekurencyjnymi sieciami neuronowymi takimi jak autoenkodery czy też sieci LSTM (ang. *long short-term memory*) [18]. Warte wyróżnienia są także opracowania naukowe wykorzystujące mgłę obliczeniową (ang. *fog computing*) [19], a także internet rzeczy (ang. *internet of things*, IoT) [20].

Pierwsze artykuły poruszające tematykę wykorzystania uczenia ze wzmocnieniem w celu sterowania ruchem powstały w połowie lat 90-tych [21]. Część z zaproponowanych rozwiązań już wtedy oferowała poprawienie wydajności względem sygnalizatorów stałoczasowych. Obecnie najpopularniejszymi stosowanymi algorytmami są te oparte o *Q-learning*, czy też bazowane na metodach aktora-krytyka (ang. *actor-critic*) [22] [23].

2.3. Zbiory danych oraz symulatory

W istniejących opracowaniach autorzy korzystają z rozmaitych narzędzi. Do najprzydatniejszych z nich należą z pewnością symulatory, których poprawne działanie oraz możliwość symulacji i wizualizacji ruchu drogowego pozwalają na naoczną weryfikację tworzonych systemów. Istnieje duża liczba różnych symulatorów, wśród których, występujących w publikacjach, można wymienić na przykład Aimsun [24], Matlab[25], Paramics[26], SUMO[27], VISSIM[28]. W pracach szczególnie często występują ostatnie dwa z nich [22]. Symulator SUMO (ang. Simulation of Urban Mobility) jest stosunkowo prostym narzędziem otwartoźródłowym pozwalającym na symulowanie obszarów miejskich. Charakteryzuje się umiarkowaną realistycznością ruchu drogowego. VISSIM (niem. Verkehr In

Städten - Simulationsmodell) jest programem komercyjnym z możliwością wyświetlania symulowanego środowiska zarówno w dwóch, jak i trzech wymiarach. Ruch pojazdów uzyskiwany za pomocą tego symulatora jest uznawany za bardziej rzeczywisty [29].

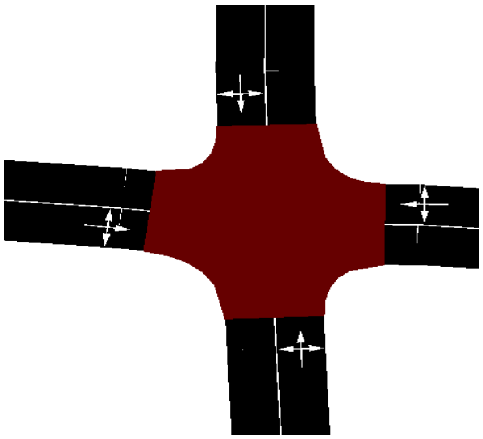
W badaniach naukowych dotyczących ruchu drogowego wykorzystuje się różnorodne metody generowania danych. Obecnie ruch drogowy jest ściśle monitorowany zarówno w celu zapewnienia bezpieczeństwa, jak i skutecznego zarządzania, planowania oraz poprawiania infrastruktury. Dostępnych jest wiele źródeł dostarczających dane o ruchu. Obejmują one zarówno instytucje rządowe, jak na przykład *Department for Transport* w Wielkiej Brytanii, jak i źródła prywatne, np. *TomTom Traffic Stats*[30] a także otwarto-źródłowe: *Telraam*[31] oraz naukowe [32]. Dane te otrzymywane są za pomocą różnych urządzeń, spośród których można wymienić między innymi kamery drogowe, pętle indukcyjne oraz smartfony wyposażone w GPS. Dzięki dużej dostępności danych, możliwe jest uzyskanie wiarygodnego ruchu ulicznego w celu weryfikacji zaprojektowanej infrastruktury. Istnieją również polskie publiczne dane, jak na przykład serwis przedstawiający wyniki pomiarów przepływu pojazdów na warszawskich drogach [33], najczęściej jednak sposób ich zbierania oraz tego, czy są udostępniane zależy od woli odpowiednich władz samorządowych.

2.4. Skrzyżowania w Polsce

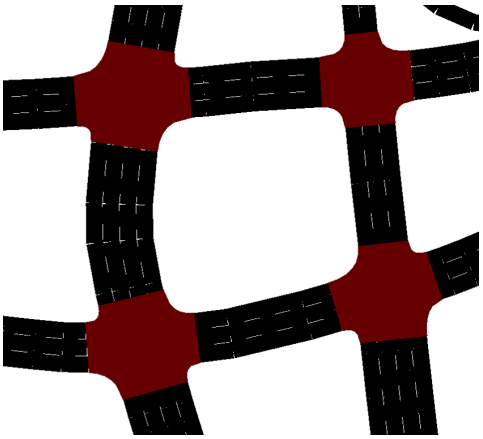
W Polsce, skrzyżowania powinny być projektowane zgodnie z wytycznymi zawartymi w dokumentach WR-D-30, w szczególności WR-D-31-(1-3) [34]. Zawierają one wytyczne rekomendowane do stosowania podczas projektowania skrzyżowań przez ministra właściwego do spraw transportu. Ich stosowanie jest zalecane, ale nie zawsze wymagane, ponieważ w wielu przypadkach warunki lokalne nie pozwalają na pełne dostosowanie się do zaleceń. Każde skrzyżowanie jest projektowane w innej lokalizacji, wraz z indywidualnym kontekstem miejscowym, do którego zalicza się między innymi obecne zagospodarowanie okolicy, warunki gruntowo-wodne czy też konieczność ograniczenia oddziaływania na środowisko. Mianem trudnych warunków określa się specyfikę terenową uniemożliwiającą zastosowanie zalecanych rozwiązań lub takie, dla których koszty zastosowania standardowych rozwiązań byłyby znacznie wyższe względem skrzyżowań standardowych.

Ze względu na sposób organizacji ruchu, skrzyżowania dzieli się na te bez, oraz z sygnalizacją świetlną. W pierwszym przypadku regulowanie ruchu odbywa się na nich przy pomocy znaków pierwszeństwa przejazdu lub zasad ruchu drogowego - oraz na skrzyżowania z sygnalizacją świetlną.

Skrzyżowania można także skategoryzować na trzy kategorie, ze względu na wymagania techniczne i użytkowe. Skrzyżowania zwykłe nie posiadające wyspy dzielącej kierunki ruchu ani środkowego pasa dzielącego. Skrzyżowania skanalizowane, które zawierają co najmniej na jednym wlocie środkowy pas dzielący lub wyspę dzielącą kończącą się w pobliżu krawędzi drogi poprzecznej. W tym przypadku, powierzchnia wyłączona z ruchu może być oznakowana poziomo lub wyspa może być wyodrębniona z jezdni. Trzecim typem są ronda, które zawierają wyspę środkową, wokół której odbywa się ruch okrężny. W określonych przypadkach wyspa może być przejezdna.



Rysunek 2.1. Skrzyżowanie ulic Siennickiej i Kobielskiej w Warszawie. Przykład skrzyżowania zwykłego.



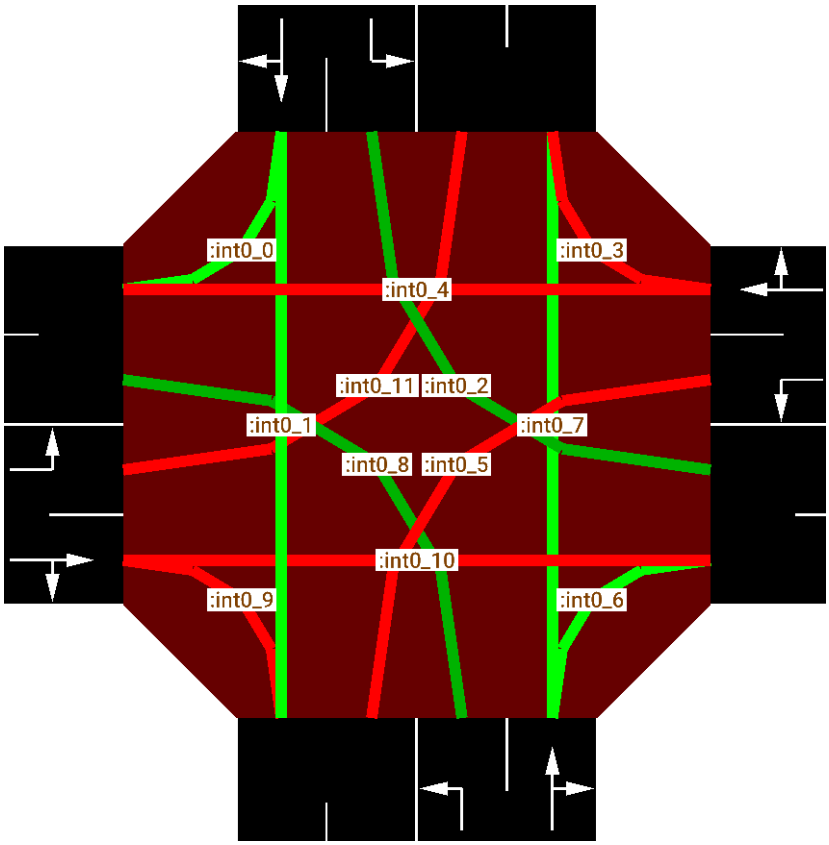
Rysunek 2.2. Rondo Jazdy Polskiej w Warszawie. Przykład skrzyżowania skanalizowanego z wewnętrznymi powierzchniami akumulacyjnymi przy wyspie centralnej dla skręcających w lewo pojazdów.

Skrzyżowania mogą mieć przeróżne układy - szczególnie skanalizowane. Dwa pierwsze typy skrzyżowań przedstawiono na rysunkach 2.1 oraz 2.2. Zrzuty ekranu przedstawionych skrzyżowań pochodzą z symulatora SUMO i zostały wygenerowane na podstawie danych z serwisu OpenStreetMap.

2.5. Sygnalizacje świetlne

Mianem sygnalizacji świetlnej określa się system mający na celu regulowanie ruchu pojazdów i pieszych tam, gdzie jest to wymagane. Składa się z kilku elementów, wśród których można wyróżnić semafor (sygnalizator), sterownik oraz urządzenia pomocnicze na przykład detektory i przyciski. W kontekście projektowania programu sterującego ruchem wyróżnić można pojęcia [35]–[37]:

- Kierunek - konkretna ustalona trasa, po której mogą poruszać się pojazdy. Pasy jezdni mogą mieć kilka kierunków.
- Faza - segment czasu w ramach całego programu, w którym określony kierunek lub ich grupa (grupa sygnałowa) ma zezwolenie na przejazd. W Stanach Zjednoczonych powszechnie są stosowane fazy NEMA (National Electrical Manufacturers Association), definiujący sposoby konfigurowania oraz zarządzania światłami [38]. W niniejszej pracy dyplomowej faza będzie stosowana zamiennie z pojęciem grupy sygnałowej.
- Cykl - całkowita sekwencja wszystkich faz programu sterowania ruchem.
- Kolizja - w kontekście sterowania ruchem określenie to nie dotyczy fizycznego zderzenia. Jest to potencjalna sytuacja, w której uczestnicy ruchu jadący w innych kierunkach mogą spowodować między sobą zderzenie lub konieczność ustąpienia pierwszeństwa innemu pojazdowi. Przykładowo lewoskręty na pasach jezdni poruszających się w przeciwnych kierunkach (np. na rysunku 2.3 dla kierunków *:int0_2* oraz *:int0_5* występuje kolizja).



Rysunek 2.3. Przykładowe skrzyżowanie czterech jezdni wraz z oznaczonymi trasami przejazdu.

- Macierz kolizji - reprezentacja, za pomocą której tworzone są programy sterowania ruchem. Pozwala w prosty sposób stworzyć bezpieczny program sterowania ruchem, unikając przy tym kolizji i maksymalizując średnią liczbę kierunków, które mogą w danym momencie jechać co bezpośrednio przekłada się na mniejsze czasy oczekiwania na skrzyżowaniu. Macierz kolizji dla skrzyżowania z rysunku 2.3 przedstawiono na rysunku 2.4. Jak można zauważyć, zgodnie z intuicją macierz kolizji jest zawsze macierzą symetryczną.
- Punkt kolizji - miejsce znajdujące się w obszarze skrzyżowania, w którym następuje przecięcie lub połączenie torów jazdy co najmniej dwóch kierunków.
- Długość kolejki - liczba samochodów oczekujących na światłach.
- Grupa sygnałowa - niekolidujący ze sobą podzbiór wszystkich kierunków, które w jednym, wspólnym momencie otrzymują lub tracą prawo przejazdu. Dla skrzyżowania z rysunku 2.3 może (nie musi) to być na przykład grupa składająca się z kierunków *:int0_0, :int0_1, :int0_6, :int0_7*. Dany kierunek może wchodzić w skład więcej niż jednej grupy.
- Strata czasu - jedna z metryk oceny warunków ruchu na skrzyżowaniach, określająca wymuszony czas oczekiwania na przejazd pojazdu przez skrzyżowanie.

Sygnalizacje można sklasyfikować ze względu na sposób wykonania programu i powtarzalności pracy na [39]:

id	int0_0	int0_1	int0_2	int0_3	int0_4	int0_5	int0_6	int0_7	int0_8	int0_9	int0_10	int0_11
int0_0	0	0	0	0	1	0	0	0	0	0	0	0
int0_1	0	0	0	0	1	0	0	0	1	1	1	1
int0_2	0	0	0	0	1	1	0	1	0	0	0	1
int0_3	0	0	0	0	0	0	0	1	0	0	0	0
int0_4	1	1	1	0	0	0	0	1	0	0	0	1
int0_5	0	0	1	0	0	0	0	1	1	0	1	0
int0_6	0	0	0	0	0	0	0	0	0	0	1	0
int0_7	0	0	1	1	1	1	0	0	0	0	1	0
int0_8	0	1	0	0	0	1	0	0	0	0	1	1
int0_9	0	1	0	0	0	0	0	0	0	0	0	0
int0_10	0	1	0	0	0	1	1	1	1	0	0	0
int0_11	0	1	1	0	1	0	0	0	1	0	0	0

Rysunek 2.4. Macierz kolizji dla skrzyżowania z rysunku 2.3. Wartość 1 w danej komórce oznacza występowanie kolizji między poszczególnymi kierunkami.

- Sygnalizację cykliczną stałoczasową - posiada ustaloną długość cyklu i poszczególnych sygnałów.
- Sygnalizację cykliczną zmiennoczasową (akomodacyjną) - ma ustaloną sekwencję faz, ale ich czas trwania jest zmienny i zależy od chwilowych charakterystyk ruchu dostarczanych z detektorów. Możliwe jest także pomijanie niektórych faz.
- Sygnalizację acykliczną - charakteryzuje się zmienną sekwencją faz. Sygnalizacja ta całkowicie zależy od aktualnego ruchu, fazy w niej są na bieżąco tworzone a ich czas trwania jest zmienny.
- Sygnalizację wzbudzaną - pracuje według układu stan ustalony - stan wzbudzony - stan ustalony, przykładowo na przejściach dla pieszych.

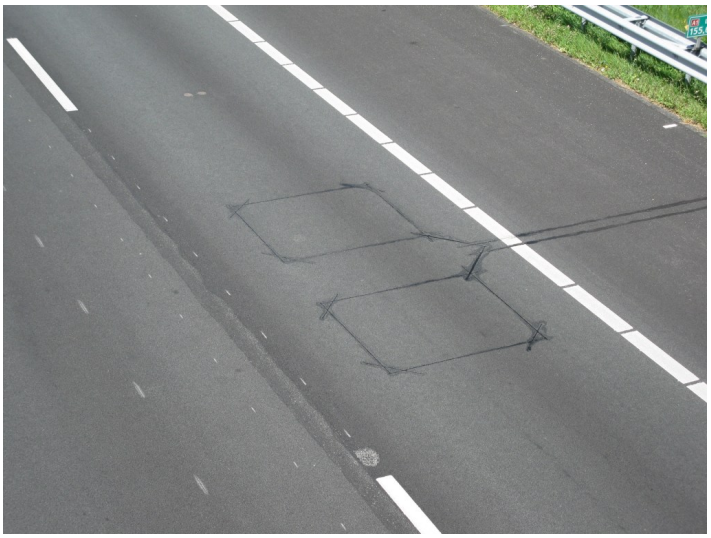
2.6. Urządzenia pomiarowe

W ramach systemów sterowania ruchem drogowym stosuje się różne rodzaje czujników. Służą one do monitorowania i zbierania długoterminowych statystyk na temat ruchu, jak i do uzyskiwania danych na temat aktualnej sytuacji w nadzorowanym obszarze. W niniejszej części przybliżono najpopularniejsze urządzenia służące do monitorowania ruchu pojazdów.

2.6.1. Pętla indukcyjna

Pętla indukcyjna to rodzaj urządzenia wykorzystywanego w systemach kontroli ruchu drogowego. Jest to detektor ruchu, który wbudowuje się w nawierzchnię drogi (*rysunek 2.5*). Zastosowania tych urządzeń obejmują zarządzanie, monitorowanie oraz zbieranie danych statystycznych ruchu drogowego. Najczęściej stosowane są na skrzyżowaniach, ale także w innych miejscach, na przykład w tunelach lub ważnych arteriach.

Pętla indukcyjna jest niezawodnym, stosunkowo prostym i bardzo odpornym na zużycie urządzeniem działającym na zasadzie indukcji elektromagnetycznej, co przekłada się na popularność ich stosowania. Składa się z przewodu elektrycznego umieszczanego w szczelinie o kształcie pętli wycinanej w powierzchni drogi, którą następnie zalewa się specjalnym materiałem. Kiedy duży, ferromagnetyczny obiekt (na przykład samochód) przemieszcza się nad pętlą, zmienia on jej indukcyjność. Taka zmiana jest następnie wykrywana przez system monitorujący, co pozwala na wykrycie obecności obiektu w pobliżu pętli. Urządzenia te konfigurowane są głównie do detekcji samochodów, dlatego może się zdarzyć, że inne, mniejsze pojazdy takie jak motocykle czy rowery nie będą wykrywane.



Rysunek 2.5. Pętle indukcyjne umieszczone w nawierzchni jezdni wykorzystywane do pomiarów prędkości przejazdu pojazdów. Źródło: [40]

Nieoczywistym sposobem zastosowania pętli indukcyjnych są urządzenia wykorzystywane do kontroli przestrzegania stosowania się do sygnalizacji świetlnej. Występują także jako element pomiarowy w dokładniejszych fotoradarach - prędkość jazdy jest wyznaczana za pomocą dwóch pętli indukcyjnych, na zasadzie odcinkowego pomiaru prędkości o małej długości odcinka [41]. Pętle indukcyjne są wykorzystywane także do sterowania sygnalizacją świetlną. Przykładowo gdy samochód zatrzymuje się nad pętlą indukcyjną na czerwonym świetle, system może wykryć obecność pojazdu i skrócić czas oczekiwania na zielone światło, poprawiając płynność ruchu.

2.6.2. Kamery

Kamery znajdują zastosowanie przy wykrywaniu wypadków, niebezpiecznych sytuacji, a także wykroczeń. Służą także do monitorowania płynności ruchu. Uzyskany obraz jest analizowany za pomocą klasycznych technik przetwarzania obrazu, lub też z wykorzystaniem uczenia głębokiego.

Dużą wadą analizy ruchu drogowego za pomocą kamer jest wysoka podatność dokładności detekcji na warunki zewnętrzne. Wyniki uzyskane z algorytmów detekcji zależą od

jakości urządzenia, różnorodności ruchu drogowego, obecności przeszkód, ale także od warunków atmosferycznych, pory dnia czy też oświetlenia.

2.6.3. Czujniki mikrofalowe

Urządzenia wykorzystujące fale elektromagnetyczne o długości mikrofalowej nazywane są czujnikami mikrofalowymi. Korzystając ze zjawiska odbicia fal radiowych pozwalają na wykrywanie obiektów oraz ich prędkości. Najpopularniejszym z punktu widzenia kierowców miejscem zastosowania tych detektorów są klasyczne fotoradary, jednak wykorzystywane są także na skrzyżowaniach oraz drogach szybkiego ruchu [42]. W przeciwieństwie do kamer wizyjnych, czujniki mikrofalowe nie są aż tak podatne na warunki pogodowe.

2.7. Inne sensory

Poza czujnikami wymienionymi w poprzednich częściach, w zależności od potrzeb zastosowanie znajdują także czujniki ultradźwiękowe, ciśnieniowe i akustyczne. Interesującym i perspektywicznym w miarę rozwoju koncepcji internetu rzeczy rozwiązaniem jest także pozyskiwaniem informacji bezpośrednio od pojazdów [43]. Dzięki połączeniu samochodów z siecią i wymianie informacji na temat położenia, prędkości, przyspieszenia i innych możliwe będzie stworzenie nowych sposobów kierowania ruchem. Z drugiej strony, sygnalizatory mogłyby przekazywać informacje na temat czasów trwania sygnałów, wpływając na poprawę zarządzania ruchem i bezpieczeństwa na drogach. Aktualnie badane są także rozwiązania korzystające z inteligentnych sygnalizacji świetlnych wraz z pojazdami autonomicznymi [44].

3. Wprowadzenie teoretyczne

W tym rozdziale zostały przedstawione zagadnienia teoretyczne wykorzystane jako punkt wyjściowy niniejszej pracy. Głównym obszarem pracy jest uczenie maszynowe, będące gałęzią sztucznej inteligencji. Przybliżono dokładniej uczenie ze wzmocnieniem (ang.*reinforcement learning, RL*) - jedną z odnóg uczenia maszynowego. W ostatnim czasie RL jest szeroko badanym rozwiązaniem dla problemu sterowania ruchem, ponieważ pozwala na stworzenie systemu potrafiącego nauczyć się zależności pomiędzy akcjami i ich skutkami w dynamicznym, zmieniającym się środowisku.

Uczenie maszynowe Uczeniem maszynowym nazywamy dziedzinę algorytmów poprawiających swoje działanie poprzez modyfikację wewnętrznych parametrów modelu matematycznego w wyniku ekspozycji na dane. W ramach tej dziedziny można wyszczególnić trzy główne nurty: uczenie nadzorowane, uczenie nienadzorowane oraz uczenie ze wzmocnieniem, którego dotyczy niniejsza praca.

Uczenie nadzorowane Korzystając z uczenia nadzorowanego dostarczamy systemowi uczonemu zarówno dane, jak i wartości wynikowe, które chcemy otrzymywać na wyjściu. Celem tego uczenia jest stworzenie modelu, który umiałby przewidzieć prawidłową odpowiedź dla danych wejściowych, z którymi jeszcze się nie zetknął. Najczęściej paradygmat ten wykorzystywany jest do problemów klasyfikacji i regresji [45].

Uczenie nienadzorowane W przeciwieństwie do uczenia nadzorowanego, w uczeniu nienadzorowanym dostarczamy modelowi dane nieposiadające uprzednio przypisanych etykiet. Dzięki temu możliwe jest wykrycie zależności między danymi, które są trudne do spostrzeżenia w inny sposób. Ten rodzaj uczenia jest wykorzystywany głównie do statystycznego przetwarzania danych (np. grupowanie, detekcja anomalii), do odkrywania powiązań dzięki nimi.

Uczenie ze wzmocnieniem Uczenie ze wzmocnieniem korzysta z odmiennych reguł względem pozostałych paradygmatów uczenia maszynowego. Zamiast dostarczać modelowi zestawu danych, zapewniamy mu dostęp do środowiska (symulowanego lub rzeczywistego), wraz z możliwością interakcji i odczytywania aktualnego stanu. Nauka odbywa się poprzez ciągłą ocenę podejmowanych działań za pomocą specjalnie zaprojektowanej funkcji. W uczeniu ze wzmocnieniem występują pojęcia takie jak:

- **Agent** - podmiot mogący podejmować decyzje w środowisku. Wykonuje akcje w zależności od dostarczonej obserwacji pochodzącej ze środowiska oraz aktualnej polityki. Agent podejmuje decyzje w określonych, równych odstępach czasu, oznaczanych $t_1, t_2, \ldots$.
- **Akcja**, $a_t \in \mathscr{A}$ - decyzja podejmowana przez agenta na podstawie danego stanu środowiska. Akcje mogą mieć wartości dyskretne, albo ciągłe.

- **Polityka**, π - funkcja, według której agent wybiera akcje, zazwyczaj zrealizowana przez sieć neuronową - w takim przypadku uczeniem nazywamy proces aktualizacji jej parametrów.
- **Stan**, s_t - zmienna zawierająca wartości (dyskretne lub ciągłe) opisujące dane środowisko, wykorzystywana w procesie uczenia polityki. Dostarcza niezbędne informacje, na podstawie których agenty podejmują decyzję. Służy do dyskretnego opisu środowiska, nawet gdy jest ono ciągłe. Decyzja podejmowana jest za pomocą funkcji stanu $Q(s_t, a_t)$. Zwraca ona przewidywaną wartość przyszłych nagród, jeżeli dla stanu s_t wykonana zostanie akcja a_t.
- **Środowisko**, $\mathscr{E}$ - otoczenie, w którym operuje agent. Może być zarówno rzeczywiste, jak i symulowane. Co określony czas środowisko opisywane jest w danym momencie za pomocą stanu, pozwala to na dyskretyzację ciągłych środowisk. W zależności od zadania, liczba różnych stanów w których może znaleźć się środowisko może być zarówno skończona jak i nieskończona.
- **Nagroda**, r_t - funkcja określająca po wykonaniu kroku przez środowisko jak bardzo był on korzystny. Pożądaną cechą agenta jest możliwość przewidywania skutków swoich akcji. Powinien on także preferować przewidywane duże, zsumowane długoterminowe nagrody względem mniejszych, uzyskiwanych w krótkim terminie. Skłonność modelu do preferowania długofalowych nagród zależy od parametru dyskonta (ang. *discount rate*) oznaczanego zazwyczaj symbolem $\gamma \in (0,1)$. Parametr ten określa, jak bardzo przyszłe nagrody są preferowane, względem natychmiastowych. Wartość bliska zero oznacza skupianie się na nagrodach uzyskiwanych niezwłocznie po dokonanej akcji. Przeciwnie wartość γ bliska 1 powoduje koncentrowanie się na długoterminowych korzyściach. Niepoprawne dobranie odpowiedniej wartości parametru γ może spowodować gorsze wyniki, lub też brak stabilności stworzonego rozwiązania [46].

3.1. Q-learning

Q-learning jest popularnym algorytmem uczenia ze wzmocnieniem. Wykorzystuje się go do nauki optymalnej polityki działania agenta w danym środowisku w celu zmaksymalizowania zdyskontowanych nagród. Jednym z kluczowych elementów tego podejścia jest funkcja $Q(s,a)$ przewidująca wartość akcji a w stanie s na podstawie doświadczeń agenta. Algorytm oparty na równaniu Bellmana dokonuje uczenia na podstawie obserwowanych stanów i otrzymanych nagród. Optymalną wartość funkcji Q^* definiuje się jako maksymalną, oczekiwaną wartość nagrody, otrzymaną w wyniku podjęcia decyzji a na podstawie stanu s za pomocą polityki π: $Q^*(s,a) = \max_\pi \mathbb{E}\big[R_t | s_t = s, a_t = a, \pi\big]$. Rozwijając tę funkcję, otrzymujemy $Q^*(s,a) = \max_\pi \mathbb{E}\big[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ... | s, a, \pi\big]$, dzięki czemu można zauważyć wpływ parametru γ na fakt preferowania długofalowych nagród. Opierając się na założeniach uczenia ze wzmocnieniem korzystamy z równania Bellmana, aby wyznaczyć iteracyjnie wartość funkcji Q: $Q_{it+1}(s,a) = \mathbb{E}\big[R + \gamma \max_{a'} Q_{it}(s', a') | s, a, \pi\big]$. Człon funkcji $\max_{a'} Q_{it}(s', a')$ oznacza maksymalną wartość funkcji Q dla następnego stanu s' rozpatrując wszystkie możliwe do podjęcia w danym momencie akcje a'. Można

zaobserwować, że funkcja Q zbiega do Q^*, gdy $i\,t \to \infty$ [47]. W algorytmie *Q-learning* jako wartość funkcji Q do której dążymy w danym momencie t definiuje się jako:

$$Y_t^{Q-learning} \equiv R_{t+1} + \gamma \max_a Q(s^{'}, a^{'}) \tag{3.1}$$

Iteracyjnie, optymalna wartość funkcji Q wyznaczana jest za pomocą równości:

$$Q_{it+1}(s,a) = (1-\alpha)Q_{it}(s,a) + \alpha Y_t^{Q-learning} \tag{3.2}$$

gdzie α oznacza współczynnik uczenia (ang. *learning rate*). Jak można zauważyć, kolejne wartości funkcji Q zależą wyłącznie od poprzednich uzyskanych wartości dla par stan-akcja. W praktyce, aby uniknąć wpadnięcia funkcji Q w minimum lokalne już na początku uczenia, stosuje się stochastyczne strategie zachłanne balansujące pomiędzy eksploatacją oraz eksploracją.

Algorytm *Q-learning* jest stosunkowo prosty i efektywny, lecz sprawdza się głównie wtedy, gdy przestrzeń stanów oraz akcji ma ograniczony, stosunkowo nieduży wymiar. Ponieważ cała przestrzeń par stan-akcja przechowywana jest w pamięci w obiekcie nazywanym tabelą wypłat podejście to obarczone jest „przekleństwem wymiarowości" (ang. *curse of dimensionality*) [48]. Dlatego też, wraz ze wzrostem skomplikowania środowiska, to jest liczby stanów i akcji, eksponencjalnie zwiększa się rozmiar *Q-table*. Z tego powodu, zastosowanie *Q-learningu* jest praktycznie niemożliwe dla skomplikowanych, rzeczywistych zadań. Funkcja Q jest uczona dla konkretnych danych, bez generalizacji, co też ogranicza jej możliwość do wykorzystania w wielu przypadkach. W celu zaradzenia temu problemowi oraz zapewnieniu wymaganej możliwości generalizacji wykorzystuje się specjalną aproksymację wartości akcji a w stanie s: $Q(s,a,\theta) \approx Q^*(s,a)$. Jako funkcję przybliżającą obecnie wykorzystuje się głębokie sieci neuronowe, a parametrem θ oznacza jej wagi. Taką sieć określa się mianem *Q-network* i wykorzystuje się ją w algorytmie nazywanym *Deep Q-learningiem (DQL)* stworzonym, aby móc pokonać ograniczenia *Q-learningu* i zapewnić rozwiązania z możliwością generalizacji.

3.2. Deep Q-learning

DQL jest pierwszym zaprezentowanym algorytmem głębokiego uczenia ze wzmocnieniem [49], za pomocą którego z powodzeniem udało się nauczyć agenta polityki na podstawie danych pochodzących z symulowanego środowiska. Sieci neuronowe uczone za pomocą *Deep Q-learningu* nazywa się głębokimi Q-sieciami, (ang. *Deep Q-Networks*, (DQN)). Agenty stworzone za pomocą DQL często charakteryzują się wydajniejszym działaniem od agentów stworzonych za pomocą innych algorytmów występujących w literaturze, a także człowieka [50]. Badanym środowiskiem był zestaw gier ATARI, zaś jako wejście każdy z agentów otrzymywał jedynie obraz środowiska, bez dźwięku. Okazało się, że w większości przypadków agent DQN sprawdzał się lepiej zarówno od pozostałych agentów, jak i człowieka.

Algorytm ten jest bezmodelowy (ang. *model-free*), co oznacza, że agent uczy się bez dokładnej wiedzy na temat modelu środowiska. Skutkuje to tym, że nie wie on jakie akcje

prowadzą do jakich rezultatów. Funkcja wartości Q jest więc uczona za pomocą tak zwanej metody prób i błędów. Celem do którego dąży algorytm dla danego stanu w kroku uczącym jest definiowany podobnie jak dla algorytmu *Q-learning* przedstawionego w *równaniu 3.1*, jednak samą wartość funkcji Q wyznacza się za pomocą sieci neuronowej z wagami θ, podobnie jak i wartość celu Y_t^{DQN} do którego dąży aproksymator:

$$Y_t^{DQN} \equiv R_{t+1} + \gamma \max_a Q(s^{'}, a^{'}, \theta) \tag{3.3}$$

W odróżnieniu od algorytmu *Q-learning*, aktualizacji wag sieci dokonuje się poprzez przeprowadzenie pojedynczego kroku uczącego za pomocą metody stochastycznego spadku gradientu, minimalizując wartość L funkcji kosztu ℓ. W praktyce, zamiast pojedynczej próbki często korzysta się z pakietów danych, co pozwala przyspieszyć proces uczenia oraz zmniejszyć szum występujący w zebranych wartościach. Do zastosowań uczenia ze wzmocnieniem jako funkcję kosztu wybiera się te służące do regresji. W literaturze najczęściej występują: błąd średniokwadratowy (MSE), RMSE (pierwiastek z MSE), funkcja straty Hubera.

$$L_{it}(\theta_{it}) = \ell\big(Q(s,a,\theta_{it}),\ Y_t^{DQN}\big) \overset{\text{RMSE}}{=} \sqrt{\mathbb{E}\big(R_{t+1} + \gamma \max_a Q(s^{'}, a^{'}, \theta_{it}) - Q(s,a,\theta_{it})\big)^2} \tag{3.4}$$

W celu balansowania eksploracji oraz eksploatacji środowiska wykorzystuje się strategię stochastyczną ε-zachłanną (ang. *ε-greedy*). Podczas wyboru akcji agent z prawdopodobieństwem $1-\varepsilon$ wybiera ją (zachłannie) na podstawie maksymalnej wartości. Akcja losowa wybierana jest z prawdopodobieństwem $\varepsilon \in [0,1]$. Wartość ta jest hiperparametrem dostosowywanym w zależności od potrzeb. Zazwyczaj, na początku uczenia ε osiąga wartości bliskie 1 (przewaga eksploracji) i w miarę postępowania procesu uczenia jest stopniowo redukowana do wartości bliskich 0 (eksploatacja). Zastosowanie takiej strategii pozwala na szybszy, a także stabilniejszy proces osiągania zadowalających polityk.

Obecnie w celu zapewnienia większej stabilności procesu uczenia wykorzystuje się dwie, zamiast jednej sieci neuronowe - w odróżnieniu od oryginalnego algorytmu. Druga sieć, nazwana siecią docelową (ang. *target network*) ma identyczną architekturę co pierwsza sieć - nazwana siecią doraźną (ang. *online network*). Na początku procesu uczenia, obie sieci mają identyczne wagi. W kolejnych krokach uczących modyfikujemy jedynie wagi θ sieci doraźnej, za pomocą której podejmujemy decyzje. Rzadko, co określoną liczbę kroków τ (zazwyczaj 10^4 do 10^6 kroków) ustala się wagi sieci docelowej $\theta^- = \theta$. Sieć ta jest wykorzystywana do określania wartości przyszłych nagród, uzyskanych przez agenta w przyszłości w następstwie wyboru akcji wybranej przez sieć doraźną. Stosując tę modyfikację, w *równaniu 3.3* parametr θ ulega zamianie na θ^-, przez co oznacza się wagi sieci rzadziej aktualizowanej. Wykorzystanie dwóch sieci jednej do wyboru akcji, a drugiej do jej ewaluacji pozwala na znaczną stabilizację procesu uczenia poprzez zapewnienie niezależności celu oraz predykcji. Rzadkie modyfikowanie θ^- wprowadza opóźnienie do wyznaczania wartości danej akcji, co z kolei pozwala sieci na osiągnięcie pewnego celu, a po skopiowaniu wag na dalsze eksploatowanie. W przeciwnym przypadku, ciągłe

przesuwanie się wartości celu może doprowadzić do zjawiska „gonienia psa za własnym ogonem", opisującego brak zdolności do osiągnięcia optimum.

Kolejnym kluczowym elementem DQL jest obszerna pamięć, w której przechowuje się informacje o przebytych doświadczeniach agenta - mechanizm *experience replay*. W buforze tym przechowuje się zestawy danych (s_t, a_t, r_t, s_{t+1}), czyli w każdym kroku działania algorytmu agent zapamiętuje stan, podjętą akcję, uzyskaną nagrodę oraz kolejny zaobserwowany stan środowiska. Co określony czas agent pobiera z tego buforu losową próbkę na podstawie której dokonuje aktualizacji sieci aproksymującej funkcję Q. Podejście to było wykorzystywane początkowo aby przyspieszyć przechowywanie osiąganych nagród, jednak jego losowość w dobieraniu próbek wykorzystywanych do uczenia pozwala na usunięcie współzależności obserwacji od środowiska [51]. Zastosowanie buforu przynosi dwie korzyści. Poza ograniczeniem stopnia korelacji kolejnych zestawów uczących, co znacząco wpływa na poprawienie stabilności procesu uczenia, jego zastosowanie pozwala na większą efektywność obliczeniową. Dzięki wykorzystaniu bufora możliwe jest wielokrotne użycie jednego zestawu danych. Dodatkowo, co ważniejsze, umożliwia zastosowanie kilkunastu (lub więcej) zestawów danych (ang. *mini-batch*) w ramach jednego kroku uczącego, co ogromnie poprawia wydajność obliczeniową algorytmu, zwłaszcza jeżeli w procesie treningu wykorzystywane jest GPU.

Listing 1: Algorytm DQL z buforem doświadczeń na podstawie artykułu [51].

Stwórz bufor D o rozmiarze N.
Stwórz funkcję aproksymującą Q o losowych wagach θ.
for $epizod = 1, M$ **do**
    Uzyskaj ze środowiska jego stan s_1.
    for $t = 1, T$ **do**
        Z prawdopodobieństwem ε wybierz losową akcję a_t, w przeciwnym przypadku $a_t = \max Q^*(s, a, \theta)$.
        Wykonaj akcję a_t i uzyskaj nagrodę r_t oraz następny stan s_{t+1}.
        Zapisz (s_t, a_t, r_t, s_{t+1}) w D.
        Pobierz próbkę zmian stanów z D.
        $y_j = \begin{cases} \mathrm{r}_j & \text{jeżeli stan j+1 jest stanem kończącym.} \\ \mathrm{r}_j + \gamma \max_{a'} \mathrm{Q}(\mathrm{s}_{j+1}, a_{j+1}, \theta^-) & \text{w przeciwnym przypadku.} \end{cases}$
        Oblicz błąd za pomocą funkcji ℓ i dokonaj aktualizacji wag sieci θ.
        if $t\%\tau == 0$ **then**
            $\theta^- = \theta$

Deep Q-learning jest stosunkowo prostym podejściem w ramach uczenia ze wzmocnieniem. Pseudokod przedstawiający główne kroki tego algorytmu przedstawiono w *listingu 1*. W praktyce, ostateczny wygląd algorytmu często zależy od konkretnego zadania. Przykładowo rzadko kiedy zdarza się, aby wagi sieci były aktualizowane co krok środowiska, o wiele częściej spotyka się uczenie co określoną liczbę kroków, wraz z zastosowaniem *mini-batchy*. Kolejną często występującą różnicą, odwołując się raz jeszcze do przykładu gier ATARI, jest występowanie stanu określającego koniec epizodu, oznaczający koniec

poziomu, przegraną lub wygraną w grze itp. Stan kończący (ang. *terminal state*) przenosi dodatkowe informacje i często jest powiązany z odpowiednim wynagrodzeniem (lub karą) w funkcji nagród. W praktyce, dane zagadnienie może nie posiadać wcale takiego stanu. Mówimy wtedy, że jest to zadanie na nieskończonym horyzoncie (ang. *continuous problem*) w przeciwieństwie do zadań epizodycznych (ang. *episodic problem*) nie posiadają one zakończenia, mogą trwać w nieskończoność. Przykładem takiego zadania może być na przykład sterownik kontrolujący poziom wody w zbiorniku retencyjnym.

Z powodu występowania funkcji max w kroku uczącym DQL, agent często uczy się nierealistycznie wysokich wartości funkcji Q. Powoduje to wybieranie akcji o zawyżonych wartościach względem innych. Występowanie nadmiernych estymacji jest szczególnie niepożądane, gdyż funkcja Q zwraca oczekiwaną skumulowaną nagrodę po podjęciu danej akcji. Niedokładne określenie oczekiwanej sumy nagród ma bezpośredni wpływ zarówno na jakość nauczonych polityk, jak i sam proces uczenia. Bezpośrednimi skutkami opisanego procesu mogą być na przykład mniejsza stabilność procesu uczenia, wolniejszy jego przebieg lub też brak zbieżności do optymalnego rozwiązania.

3.2.1. Catastrophic forgetting

Jednym z czynników, które negatywnie wpływają na stabilność procesu uczenia się sieci w ramach uczenia głębokiego jest ten opisywany w literaturze mianem *catastrophic forgetting*[52]. Występuje on szczególnie wtedy, gdy zbiór danych uczących nie jest stały, np. w ramach uczenia przyrostowego, gdy próbujemy dostosować istniejący model do nowego zadania. Podobnie w ramach uczenia ze wzmocnieniem dane, które uzyskujemy w początkowych, losowych i częściowo losowych krokach systemu ulegają stopniowemu nadpisywaniu przez nowsze. Jeżeli uczenie będzie trwało odpowiednio długo, a agent będzie stale douczany, istnieje duże prawdopodobieństwo wystąpienia zjawiska zatracenia umiejętności optymalnego (lub też wystarczająco dobrego) reagowania na dany stan wejściowy. W skrajnym przypadku, jeżeli agent stanie się niezwykle dobry w wykonywaniu swojego zadania, możliwa jest sytuacja, że w buforze, w którym przechowywane są dane uczące, znajdą się bardzo podobne do siebie stany. Wtedy też, odpowiadające im nagrody również będą zbliżone względem siebie. W wyniku tego możliwe jest, że sieć neuronowa zostanie przeuczona, co doprowadzi do znacznego spadku jakości działania agenta.

Jednym ze sposobów na walkę ze zjawiskiem zapominania poprzednio nauczonego poprawnego działania jest zastosowanie opisanego wcześniej buforu *experience replay*. Pozwala on na zmniejszenie skorelowania następujących po sobie danych, które otrzymuje uczona sieć. Bez jego zastosowania modyfikacje wag sieci zależałyby od kolejno występujących w środowisku stanów. Są one wysoce ze sobą powiązane, ponieważ pomiędzy kolejnymi krokami środowiska jego stan nie ulega dużym zmianom. Innym następstwem wykorzystania bufora jest to, że do uczenia sieci wykorzystuje się doświadczenia uzyskane nie tylko za pomocą najnowszej polityki.

Określenie odpowiedniego rozmiaru pamięci jest zadaniem wymagającym przeprowadzenia strojenia, analogicznie do innych hiperparametrów. Zbyt duży jej rozmiar spowoduje wyczerpanie dostępnej pamięci sprzętowej. Za mały z kolei może doprowadzić

do wystąpienia opisywanego zapominania. Losowe próbki pochodzące z początkowego uruchomienia zostaną nadpisane przez nowsze. Wraz z rośnięciem skłonności strategii zachłannej do eksploatacji maleje także prawdopodobieństwo uzyskania cennych stanów uzyskanych za pomocą eksplorujących akcji. W konsekwencji może to doprowadzić do przeuczenia sieci, oraz w ekstremalnych sytuacjach do rozbieżności algorytmu [53]. Często stosowaną modyfikacją bufora jest ta pozwalająca na uchronienie przed nadpisywaniem części pamięci. Przykładowo zachowanie początkowych 10% stanów, podczas których agent miał duże tendencje od eksploracji pozwoli na przypominanie, jakich akcji nie podejmować.

3.2.2. Curriculum learning

Curriculum learning (ang. *curriculum* - program nauczania) to metodyka wyboru kolejności i doboru trudności zadań, jakim poddawany jest agent tak, aby możliwie najbardziej przyspieszyć jego uczenie oraz jakość działania [54] [55]. Korzysta ona z intuicyjnego dla ludzi podejścia, w którym uczniowi przedstawia się trudniejsze zadania, dopiero po tym gdy rozwiąże i zrozumie zasady stojące za łatwiejszymi.

Rozpoczęcie nauki na prostych przykładach lub też korzystając z uproszczonego środowiska pozwala na szybsze osiągnięcie dostatecznych wyników, zarówno pod kątem wydajności obliczeniowej jak i sposobu działania, ponieważ mniej czasu jest marnowane na zaszumione czy też trudne dane, z którymi uczący nie może sobie jeszcze poradzić. W następnych krokach powoli dostarcza się bardziej skomplikowane zadania, w których uczący może skorzystać ze zgromadzonej wiedzy. Takie podejście pozwala sieci na lepsze „zrozumienie" środowiska, co bezpośrednio przekłada się na lepszą skuteczność rozwiązania [56].

3.3. Double DQN

W algorytmie DQN, jednym z większych problemów jest tendencja do zawyżania wartości funkcji Q (ang. *overestimation bias*). W momencie aktualizacji wag sieci wartość docelowa funkcji wyznaczana jest jako suma otrzymanej nagrody oraz maksymalnej wartości tej funkcji dla następnego stanu obniżonej o współczynnik γ, a więc sieć uczona jest tak, aby wybierać akcję o najwyższej wartości. Z powodu takiego sposobu oceny, szczególnie na początku procesu uczenia gdy wagi sieci są ustawione losowo oszacowania wartości funkcji Q mogą być znacznie przeszacowane. Prowadzi to do systematycznego błędu w kierunku akcji niesłusznie ocenionych jako korzystne i wybranych za pomocą operacji max. Może to skutkować wyborem nieoptymalnych akcji, nadmierną eksploatacją pewnych akcji a także wprowadzić niestabilność do procesu uczenia [57].

Aby przeciwdziałać problemowi zawyżonych wartości funkcji Q, została przedstawiona modyfikacja DQN powstała z połączenia algorytmów Double Q-learning oraz DQN, nazwana *Double DQN* (DDQN) [58]. W odróżnieniu od klasycznego DQN, w DDQN zarówno sieć doraźna jak i docelowa są wykorzystywane do obliczania docelowych wartości funkcji Q podczas aktualizacji sieci. Ma to na celu oddzielenie od siebie operacji max w wyborze akcji oraz jej ewaluacji. Dlatego też pierwsza sieć o wagach θ wykorzystywana jest do

wybierania akcji dla danego stanu. Druga sieć - o identycznej architekturze i wagach oznaczanych θ^- - określa wartość wybranej akcji w danej sytuacji. Funkcja celu dla algorytmu DDQN definiowana jest jako:

$$Y_t^{DDQN} \equiv R_{t+1} + \gamma Q(s^{'}, \text{argmax}Q(s^{'}, a^{'}, \theta),\ \theta^-) \tag{3.5}$$

Jak można zauważyć w *równaniu 3.5*, wybór akcji dla danego stanu odbywa się analogicznie jak z wykorzystaniem algorytmu DQN, jedyną różnicą w funkcji celu jest ocena podjętych decyzji za pomocą sieci neuronowej o wagach θ^-. Sieć celu aktualizuje swoje wagi poprzez kopiowanie wag θ cyklicznie, zgodnie z ustalonym hiperparametrem. Obecnie w literaturze często występuje także metoda powolnego zmierzania wag θ^- do θ, nazwana *soft update* [59]. Za jej pomocą wagi funkcji celu powolnie (w zależności od parametru τ) zmierzają do osiągnięcia zamierzonej wartości zgodnie z iteracyjnym wzorem $\theta^- = \tau\theta + (1-\tau)\theta^-$, gdzie $\tau << 1$.

Algorytm DDQN względem DQN cechuje się dokładniejszym szacowaniem wartości funkcji Q. Dzięki temu charakteryzuje się większą stabilnością uczenia oraz szybszym zbieganiem polityk do optymalnych. Z drugiej strony, algorytm DQN dobrze sprawdza się dla mniej skomplikowanych zadań, jest prostszy w implementacji i szybszy. Wynika to z występowania dwóch sieci w algorytmie DDQN. Agenty korzystające z polityk uzyskanych za pomocą *Double Q-learningu* osiągają przynajmniej tak dobre wyniki w testach jak ci stworzeni za pomocą DQN. Dzięki mniejszej skłonności do przeszacowywania wartości funkcji Q agenty są też bardziej odporne na zaszumione dane wejściowe.

3.4. Inne metody uczenia ze wzmocnieniem

Poza wymienionymi w poprzednich częściach metodami istnieje wiele innych podejść i ulepszeń tych technik. Wśród nich występują metody modyfikujące połączenia w sieci neuronowej, zmieniające mechanizm działania buforu pamięci czy też korzystające z wielokrotnych kroków w przyszłości zamiast pojedynczej nagrody. Każde z rozszerzeń wnosi coś nowego do podstawowego konceptu DQN lub korzysta z pomysłów przedstawianych w innych pracach [60]. Większość źródeł za cel obiera poprawienie wydajności, dokładności i stabilności procesu uczenia względem istniejących rozwiązań.

4. Narzędzia

W celu stworzenia zaplanowanego systemu konieczne było określenie potrzebnych do jego realizacji narzędzi. Wybrano symulator tworzący środowisko, w którym operował agent, dokonano wyboru odpowiedniego języka programowania oraz wyszukano biblioteki wykorzystane do zaimplementowania rozwiązania. Zostały określone także pomniejsze biblioteki służące do manipulacji oraz wizualizacji danych.

4.1. Symulator

Wybór odpowiedniego symulatora w uczeniu ze wzmocnieniem odgrywa kluczową rolę. Odpowiednio dobrany symulator musi dobrze odzwierciedlać rzeczywiste warunki tak, aby uzyskane umiejętności były przenośne i użyteczne. Dodatkowo, symulator powinien być wystarczająco złożony, aby umożliwić agentowi doświadczanie różnorodnych sytuacji i uczenie dostosowywania się do zmieniających warunków. Ponadto, powinien on być łatwy w obsłudze dla użytkowników umożliwiając intuicyjne korzystanie przy jednoczesnym udostępnianiu API tak, aby możliwie maksymalnie zautomatyzować długotrwały proces uczenia oraz testowania. Poprawny wybór symulatora ma istotny wpływ na efektywność uczenia ze wzmocnieniem, dlatego też decyzja ta powinna być starannie przemyślana. Spośród występujących w literaturze symulatorów zdecydowano się na wykorzystanie symulatora SUMO [27]. Wybór ten był motywowany posiadaniem wyżej wymienionych cech, jak i zgromadzonej wokół niego społeczności, co pozytywnie przekłada się na rozwiązywanie napotkanych problemów.

Symulator SUMO to oprogramowanie o otwartym kodzie źródłowym służące do symulacji ruchu drogowego. Jest to projekt rozwijany od 2001 roku przez grupę badawczą z Niemieckiej Agencji Kosmicznej oraz społeczność użytkowników. SUMO służy głównie do modelowania ruchu drogowego w miastach, ale może być wykorzystywany także do symulowania ruchu na obszarach wiejskich lub drogach ekspresowych i autostradach.

Do najważniejszych zalet tego symulatora należy jego wszechstronność, wysoka elastyczność i możliwość dostosowywania scenariuszy testowych do własnych potrzeb. Interakcje pomiędzy nie są pomijane pojazdami, co przekłada się na lepsze modelowanie rzeczywistych warunków drogowych. W prosty sposób możliwe jest też stworzenie modeli prawdziwych siatek ulic włącznie ze skrzyżowaniami o nietypowych kształtach.

Szczególnie ważną cechą symulatora jest jego możliwość symulacji sensorów ruchu drogowego, które występują w rzeczywistym świecie. Symulowane czujniki umożliwiają dokładne modelowanie i analizę różnych aspektów ruchu drogowego. Z punktu widzenia projektu, najprzydatniejszymi urządzeniami pomiarowymi były pętle indukcyjne oraz kamery.

Z punktu widzenia projektu tworzonego w ramach pracy dyplomowej najważniejszą cechą SUMO jest możliwość jego integracji ze środowiskami programistycznymi. Za pomocą API dostarczonego razem z symulatorem w prosty sposób można sterować każdym elementem symulacji, od pojazdu po aktualny stan sygnalizacji świetlnej. Dostępnych jest również wiele narzędzi ułatwiających prace nad projektem jak na przykład OSM Web

Wizard [61], pozwalający na przetworzenie fragmentu mapy z serwisu OpenStreetMap na siatkę ulic symulatora. Równie intuicyjne jest także generowanie ruchu ulicznego.

4.2. Biblioteka do uczenia maszynowego

Aktualnie, najpopularniejszymi bibliotekami wykorzystywanymi do uczenia maszynowego są TensorFlow [62] oraz PyTorch [63]. Do stworzenia projektu w ramach pracy dyplomowej zdecydowano się na użycie biblioteki PyTorch. Jest to szybka i stosunkowo prosta w użyciu otwartoźródłowa biblioteka napisana w językach Python i C++. W porównaniu z TensorFlow, charakteryzuje się łatwą możliwością wykorzystania GPU i prostszym procesem debugowania. Najważniejszymi zaletami TensorFlow są lepsze narzędzia do wizualizacji i monitorowania procesu uczenia. Obie biblioteki mają rozbudowaną społeczność oraz wiele materiałów wprowadzających do ich użycia.

TensorFlow jest częściej zastosowywany w wielkoskalowych projektach wymagających maksymalnej wydajności, podczas gdy Pytorch wykorzystywany jest w pracach naukowych, takich jak niniejsza, głównie z powodów elastyczności, prostoty w obsłudze i rozwiązywaniu problemów związanych z implementacją czy użytkowaniem.

5. Projekt

W tej części pracy przedstawiono przebieg fazy projektowej przeprowadzonej w ramach dyplomowej pracy magisterskiej, rozpoczynając od stworzenia prototypu. Na jego podstawie wyciągnięto kluczowe wnioski dotyczące występujących w nim ograniczeń oraz możliwych sposobów ich ominięcia. Pomysły te posłużyły jako fundament do zaprojektowania i opracowania ostatecznego rozwiązania.

5.1. Prototyp

Pierwszym krokiem podczas wykonywania projektu, było stworzenie prototypu końcowego rozwiązania. Pierwsza wersja miała za zadanie sterowanie światłami na pojedynczym skrzyżowaniu czterech trzypasmowych dróg. W tym celu stworzono proste rozwiązanie, w którym pojedynczy agent zajmował się kontrolowaniem wszystkich możliwych kierunków przejazdu w ramach skrzyżowania. Porównując prototyp do innych występujących w literaturze rozwiązań, a także do finalnej wersji projektu stwierdzono, że charakteryzował się on zarówno małą liczbą danych dostarczanych agentowi w formie stanu, jak i prostą siecią neuronową złożoną z dwóch w pełni połączonych warstw.

Sterowanie ruchem uzyskane za pomocą prototypu było o wiele gorsze pod względem czasu oczekiwania na przejazd zarówno od końcowej wersji rozwiązania, jak i od prac innych autorów, jednakże praca nad prototypem pozwoliła na bliższe przyjrzenie się sposobowi w jaki zachodzi sterowanie ruchem. Mimo niskiej jakości uzyskanego sterowania, sam proces pracy nad prototypem miał nieoceniony wkład w proces projektowania ostatecznego kształtu systemu. Najważniejszymi efektami wstępnej fazy projektowej było dogłębne zapoznanie się z zagadnieniem od strony praktycznej oraz zidentyfikowanie słabych i mocnych stron zastosowanego podejścia. Dzięki bezpośredniej styczności z problemem sterowania ruchem, możliwe było zaradzenie poznanym ograniczeniom prototypu.

W odniesieniu do prototypu oraz innych prac głównym ograniczeniem zastosowanych rozwiązań był brak elastyczności w kontekście układu skrzyżowania. Z powodu zarówno zastosowanych technik, jak i danych, które były dostarczane agentom w postaci wejść, niemożliwe było zastosowanie jednego rozwiązania na dwóch różnych skrzyżowaniach, nawet jeżeli różniły się wyłącznie liczbą pasów na drogach. Takie ograniczenie powodowałoby konieczność dostosowania i przebudowy całego rozwiązania, w zależności od układu dróg dochodzących do skrzyżowania.

5.2. Określenie założeń pracy

Dzięki dokładnemu zbadaniu ograniczeń prototypu zdefiniowano kluczowe założenia i ograniczenia, które obowiązywały w dalszym przebiegu projektu.

5.2.1. Sieć dróg

W ramach niniejszej pracy, przeprowadzane rozważania skoncentrowane były wyłącznie na ruchu kołowym, ze świadomym pominięciem ruchu pieszego oraz szynowego. Pod względem struktury dróg nie określono dokładnych ograniczeń dotyczących liczby pasów,

jednakże w ramach stworzonych siatek nie przewidziano drogi szerszej, niż o czterech pasach. Skupiono się na skrzyżowaniach o co najwyżej pięciu drogach wlotowych. Ograniczenie górne liczby wlotów wynika z tego, że w wielu miastach ponad 99,9% skrzyżowań to skrzyżowania trzy-, cztero- lub pięciowlotowe [64], co pozwoliłoby obsłużyć znakomitą większość przypadków skrzyżowań występujących na świecie.

Pomimo zastosowanych ograniczeń dotyczących siatki dróg, ostateczne rozwiązanie jest możliwe do zastosowania zarówno na skrzyżowaniach o większej liczbie pasów, jak i większej liczbie dróg wlotowych.

5.2.2. Pojazdy

Założono, że wszystkie pojazdy uczestniczące w ruchu będą identyczne pod względem długości, maksymalnej prędkości i innych charakterystyk ruchu. Taki wybór pozwolił na ograniczenie wpływu losowości na proces rozwoju projektu, oraz skoncentrowanie się na optymalizacji systemu.

5.2.3. Środowisko

Zdecydowano, że wszystkie sygnały będą miały zdefiniowany minimalny czas trwania, czyli po włączeniu danego światła nie była możliwa jego zmiana przed upływem określonego czasu. Było to podyktowane zarówno oszczędnością obliczeniową, jak i analogicznymi rozwiązaniami zastosowanymi w świecie rzeczywistym oraz pracach naukowych. Decyzja ta wynika z faktu, iż od momentu zapalenia się zielonego światła do zauważenia sygnału przez oczekującego kierowcę, oraz uzyskania dostatecznej do przejazdu prędkości mija od kilku do kilkunastu sekund.

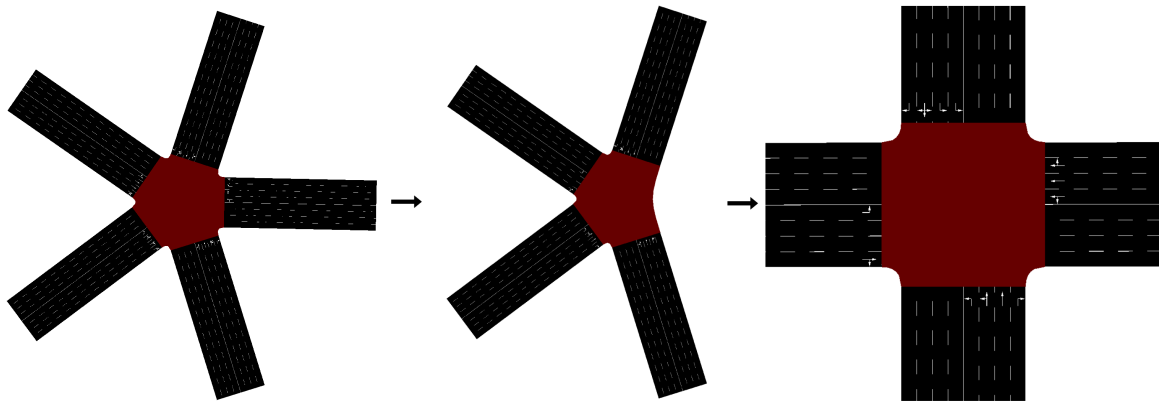
Pod kątem informacji uzyskiwanych ze środowiska, dane dostarczane agentom były realistyczne. Oznacza to, że pola stanu, na podstawie których podejmowane były decyzje, były takie, które rzeczywiście można uzyskać za pomocą dostępnych czujników i technologii. Podejścia algorytmiczne korzystają z prostych danych wejściowych takich jak długość kolejki pojazdów czy czas oczekiwania na zielone. Część prac stosujących uczenie ze wzmocnieniem wykorzystują takie dane wejściowe jak dokładne położenia i prędkości samochodów na pasach dojazdowych [22] [65], co aktualnie nie może mieć miejsca przez brak dostatecznie rozwiniętych technologii komunikacji pojazdów z systemami kontroli ruchu.

5.3. Faza projektowa

Największą rozpoznaną wadą rozwiązania agent-skrzyżowanie był brak możliwości zastosowania raz nauczonego agenta na nawet minimalnie różniących się skrzyżowaniach. Aby temu zapobiec, rozpatrzono kilka podejść rozwiązania następującego problemu. W tej części pracy zdecydowano się przedstawić dwa najbardziej wartościowe i nadające się do zrealizowania.

Pierwszym pomysłem była przebudowa danych wejściowych. W tym celu rozważano stworzenie agenta potrafiącego obsłużyć hipotetyczne, pięcio-wlotowe, duże pod względem liczby pasów skrzyżowanie. Następnie dostosowywano jego wejścia w zależności od

sytuacji w jakiej miałby być wykorzystany. Polegało to na ręcznym ustawieniu sygnałów wejściowych na wartość zerową dla tych pasów, które w rzeczywistości nie występowały. Przykładowe kroki dostosowywania skrzyżowania przedstawiono na *rysunku 5.1*. Takie podejście pozwoliłoby na rozwiązanie problemu braku elastyczności rozwiązania, jednak wraz z nim powiązana byłaby konieczność uprzedniego zdefiniowania znacznej liczby faz - po jednej dla każdego pasa - co mogłoby spowodować brak elastyczności agentów w tworzeniu programu sygnałów świetlnych. Oznaczałoby to z jednej strony zwiększanie poziomu abstrakcji tego rozwiązania a z drugiej dodanie kolejnych cech ją ograniczających. Uczenie takiego rozwiązania byłoby także niezwykle czasochłonne, ponieważ wymagane stałoby się pokrycie w scenariuszach uczących każdego wariantu „zaślepiania" pasów, czyli ustawiania ich wartości na zero. Z powodu opisanych ograniczeń oraz braku perspektyw na ich obejście porzucono ten pomysł i spróbowano rozwiązać znalezione problemy w inny sposób.



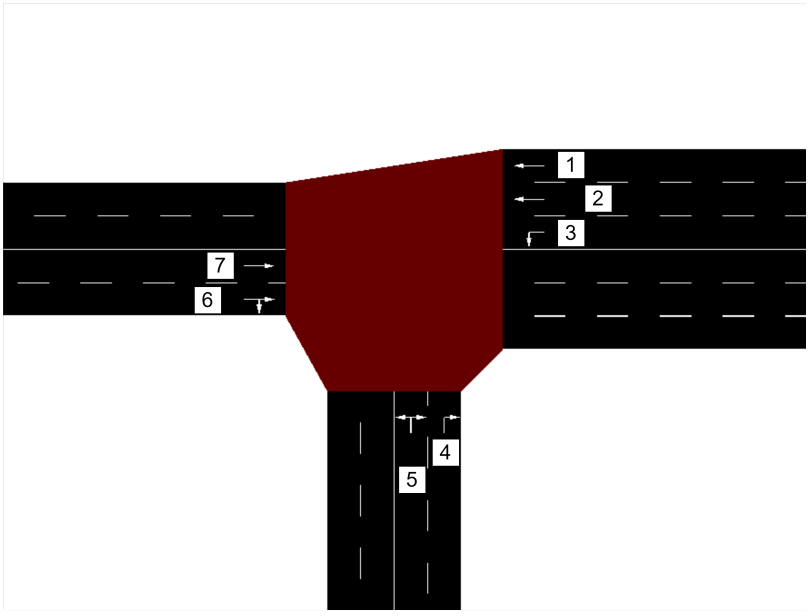
Rysunek 5.1. Koncepcja agenta-pasa. Kolejne kroki dostosowania skrzyżowania pięciowlotowego do czterowlotowego. Z punktu widzenia agenta, każde ze skrzyżowań posiadałoby tę samą strukturę. Z punktu widzenia użytkownika możemy zastosować jedno rozwiązanie do wszystkich skrzyżowań.

W ramach kolejnej koncepcji postanowiono skorzystać z wieloagentowego uczenia ze wzmocnieniem (ang. *Multi-Agent Reinforcement Learning, MARL)*. Jest to podejście obecnie często badane w pracach opisujących zastosowanie sztucznej inteligencji w celu sterowania sygnalizacją świetlną prowadząc do usprawnienia ruchu drogowego [66] [67] [68] [69]. Pomimo iż przeróżne źródła opisują badanie wykorzystanie MARL, żadne z napotkanych nie porusza zagadnienia stworzenia uniwersalnego agenta. Przytaczane prace z powodzeniem ograniczały czas konieczny na przejazd przez światła, także w przypadku skomplikowanych siatek ulic, koncentrując się głównie na maksymalizacji przepustowości. Jednak tematyka uniwersalności oraz możliwości zastosowania przedstawianych rozwiązań w rzeczywistych warunkach była pomijana lub poruszana jedynie pobieżnie. Większy nacisk położono na efektywność systemów, a mniejszy na ich adaptacyjność i uniwersalność. Pod względem wewnętrznego mechanizmu działania zaś, prace te działały podobnie do komunikujących się między sobą instancji prototypu. Problemy napotykane podczas tworzenia systemu z pojedynczym agentem stają się jeszcze bardziej istotne i skomplikowane w systemach wieloagentowym [70]. Z tego też powodu konieczny jest dłuższy czas poświęcony na staranne zaprojektowanie rozwiązania.

Z tego powodu, aby ograniczyć zróżnicowanie występujące w systemie, głównym elementem niniejszego wieloagentowego podejścia była homogeniczność agentów pod względem zarówno wymiaru danych wejściowych, jak i wykorzystywanej sieci. W odróżnieniu od uprzednio opisywanej koncepcji agenta-skrzyżowania, zdecydowano się na powiązanie agenta z pasem wlotowym do skrzyżowania. Takie podejście nazwano agent-pas. Zgodnie z założeniem, agent-pas miał być możliwy do zastosowania na każdym skrzyżowaniu po uprzednim nauczeniu. W odróżnieniu od agenta-skrzyżowanie, szczególną zaletą niniejszej koncepcji był brak konieczności szczególnego dostosowywania agenta w zależności od skrzyżowania, na którym miałby być wykorzystany.

5.4. Agent-pas

Z powodu znacznego zwiększenia liczby agentów w ramach każdego skrzyżowania, a także z powodu ograniczenia stopnia widoczności globalnego stanu, konieczne stało się staranne zaprojektowanie zasad wymiany informacji pomiędzy agentami. Kluczowe znaczenie miał także dobór odpowiednich danych wejściowych. Musiały dostarczać wystarczających informacji na temat aktualnego stanu niezbędnych do zarządzania ruchem. Z drugiej strony, wraz ze wzrostem wymiaru danych wejściowych, znacznie rosły wymagane zasoby sprzętowe potrzebne do przeprowadzenia symulacji oraz procesu uczenia. Szczególny wpływ miało to dla bardziej skomplikowanych siatek ulic - wśród badanych scenariuszy potrzebnych było ponad 100 agentów. Przykładowe zastosowanie koncepcji agenta-pasa przedstawiono na *rysunku 5.2*. Ważnym punktem zastosowanego podejścia jest identyczność pod względem budowy wewnętrznej wszystkich agentów obsługujących różne kierunki ruchu pojazdów.



Rysunek 5.2. Koncepcja agenta-pasa. Agent został zaprojektowany tak, aby możliwe było zastosowanie tego samego rozwiązania dla dowolnego skrzyżowania. Na powyższym skrzyżowaniu znajduje się siedmiu agentów oznaczonych cyframi. Kontrolują światła dla każdego z pasów wlotowych.

W odróżnieniu od klasycznych rozwiązań jak na przykład sygnalizacja cykliczna lub niektórych z opracowań naukowych ważną zaletą przedstawionego podejścia było uniknięcie wymuszonego cyklu programu. Tradycyjne metody opierały się na głównie na predefiniowanych sekwencjach sygnałów. W porównaniu, rozwiązanie agent-pas cechowało się większą dynamiką i zdolnością adaptacji do aktualnych warunków, właściwie prawie całkowicie pozwalając na ciągłe modyfikowanie grup sygnałowych. Minusem tej cechy była konieczność starannego zaprojektowania funkcji nagród tak, aby nie dopuścić do zagłodzenia pozostałych agentów w ramach pojedynczego skrzyżowania.

5.4.1. Stan

Dostarczane do agentów informacje na temat stanu skrzyżowania musiały być na tyle dokładne, aby pozwalały na efektywne sterowanie ruchem pojazdów. Jednocześnie przekazywany stan nie mógł obejmować całościowego obrazu każdego skrzyżowania, ponieważ spowodowałoby to różnice w rozmiarach stanu w zależności od układu danego skrzyżowania. W wyniku tego, nie byłoby możliwe zapewnienie homogeniczności agentów, gdyż część z nich miałby różne wymiary danych wejściowych. Ponadto, znaczny rozrost wymiaru stanu mógłby nieproporcjonalnie zwiększyć złożoność i trudność w procesie uczenia sieci neuronowej. Jednocześnie konieczne było umożliwienie agentom kooperacji, szczególnie w ramach jednej drogi wlotowej, ponieważ zazwyczaj kierunki wychodzące z jednej drogi (dalej nazywane grupą) nie są ze sobą konfliktujące. Wymagane również było zapewnienie informacji na temat tego, czy kolejka danego agenta jest stosunkowo większa, czy mniejsza od pozostałych.

Aby umożliwić komunikację między agentami, jako część danych wejściowych dostarczano im przeskalowane, bezwymiarowe wartości odnoszące się do konkretnej grupy lub całego skrzyżowania. Taki wybór danych miał na celu znormalizowanie i ujednolicenie danych przekazywanych agentom, dostarczając im dane w skali względnej, niezależnej od topologii skrzyżowania. Poza informacjami powiązanymi z innymi agentami, każdy z nich otrzymywał indywidualne dane takie jak opóźnienie, obecny sygnał czy długość kolejki. Pozwoliło to na dostosowanie się światła wystawianego przez agenta do sytuacji na jego pasie drogowym.

Pojedyncza próbka danych często nie dostarcza zupełnego obrazu sytuacji, przykładowo nie pozwala uchwycić dynamiki zmian przekazywanych wartości. Dlatego też, analogicznie do innych prac zamiast pojedynczej obserwacji, jako stan dostarcza się serię następujących po sobie obserwacji [50]. Zastosowanie sekwencyjnego podejścia naruszałoby wymaganie dotyczące braku skorelowania obserwacji, jednakże dzięki zastosowaniu buforu *experience replay* korelacja między kolejnymi danymi uczącymi nie występuje [71].

Szczegółowy opis pól stanu został przedstawiony w następnym rozdziale.

5.4.2. Akcje

W literaturze występuje wiele różnych podejść do tematu przestrzeni akcji, które podejmują agenty. Do najpopularniejszych należą: wybór następnej fazy o określonym czasie

trwania[72], wybór czasu trwania kolejnej fazy zielonej w cyklu [65], przesunięcie w cyklu fazy, zmiana długości czasu trwania fazy [73] lub też różne ich kombinacje.

Zwiększanie przestrzeni akcji poprawia jego możliwość dobrego dostosowania się do aktualnego stanu, jednak zwiększa zarówno koszt obliczeniowy potrzebny w ramach jednego wyboru akcji, jak i znacznie wydłuża wymagany czas uczenia przez konieczność lepszego pokrycia scenariuszami uczącymi.

Aby zachować względną prostotę rozwiązania postanowiono aby każdy z agentów wybierał jedynie pożądany stan świateł, biorąc pod uwagę tylko kolory zielony i czerwony. Eliminacja koloru żółtego wynika z zauważonych trudności związanych z integracją tego koloru świateł do prototypu. Wykorzystanie tego koloru nie ma także tak znacznego wpływu w kontekście sterowania ruchem, jak światła czerwone i zielone. Minimalny czas trwania sygnału był ustalony, jednakże agenty mieli dowolność w jego wydłużaniu w zależności od stanu. Ograniczenie liczby możliwych do podjęcia akcji wynikało także z liczby agentów w ramach każdego skrzyżowania. Dozwolenie na znaczną liczbę różnych decyzji znacznie utrudniałoby koordynację pracy w ramach pojedynczego skrzyżowania.

5.4.3. Funkcja nagrody

Funkcja nagrody została stworzona tak, aby nagradzać i karać agenta za dokonane wybory wpływające na minimalizację kolejki na danym pasie. Niepożądanym skutkom akcji przypisano ujemne wartości, a pozytywnym wartości dodatnie. Dlatego też zadaniem wykorzystanego algorytmu uczenia ze wzmocnieniem była maksymalizacja uzyskiwanych wartości. Funkcję rozszerzono o nagradzanie za skoordynowane działanie w ramach jego grupy, przy jednoczesnym zadbaniu o niewystępowanie przesadnie długich sygnałów.

Podobnie do stanu, szczegółowy opis funkcji nagród został przedstawiony w następnym rozdziale.

6. Implementacja

W tej części pracy zostały przedstawione informacje dotyczące implementacji stworzonego rozwiązania. Opisano w nim techniczne aspekty wieloagentowego systemu, w którym pojedynczy agent stworzony za pomocą DDQN miał za zadanie kontrolować światła ruchu drogowego dla pojedynczego pasa wlotowego do skrzyżowania. Przybliżono strukturę sieci neuronowej, ogólny schemat działania procesu uczenia oraz stworzone mechanizmy, które powstały w oparciu o podjęte decyzje projektowe.

6.1. Hiperparametry

W procesie uczenia wybrane hiperparametry miały wartości przedstawione w *tabeli 6.1*. Wartości te zostały dostrojone poprzez stopniowe modyfikowanie i obserwowanie wpływu tych zmian na wyniki kilkukrotnie przeprowadzonych testów.

Tabela 6.1. Wartości hiperparametrów uzyskane w procesie strojenia

Wielkość	**Wartość**
Learning rate	0,00001
γ - Stopa dyskontowa	0,95
ϵ_{min} - minimalna wartość parametru ϵ	0,0
Liczba kroków symulacji do osiągnięcia wartości ϵ_{min}	10800
Batch size	32
Czas trwania kroku symulacji [s]	1
Minimalny czas trwania fazy [s]	5
Liczba obserwacji wchodzących w skład stanu	12
Liczba kroków symulacji do skopiowania wag $\theta^- := \theta$	50000
Liczba kroków symulacji do wykonania kroku uczącego	8
Rozmiar bufora *experience replay*	128000
Część trwałych wpisów w buforze	12800
Maksymalny czas trwania epizodu [h]	24
Liczba pomijanych kroków	25

Czas trwania kroku symulacji był równy 1 sekundzie, dlatego też liczba kroków do ustania działania strategii ϵ-zachłannej równała się trzem godzinom. Liczba pomijanych kroków oznacza, ile pierwszych kroków w każdej symulacji było wykonanych losowo, czyli bez podejmowania akcji przez agenty. Pozwalało to na wstępne przemieszczenie się pojazdów (każdy pojazd pojawiał się na krawędzi siatki ulic), zapełnienie buforów na podstawie których agenty podejmowały decyzję oraz na minimalne skrócenie czasu trwania symulacji.

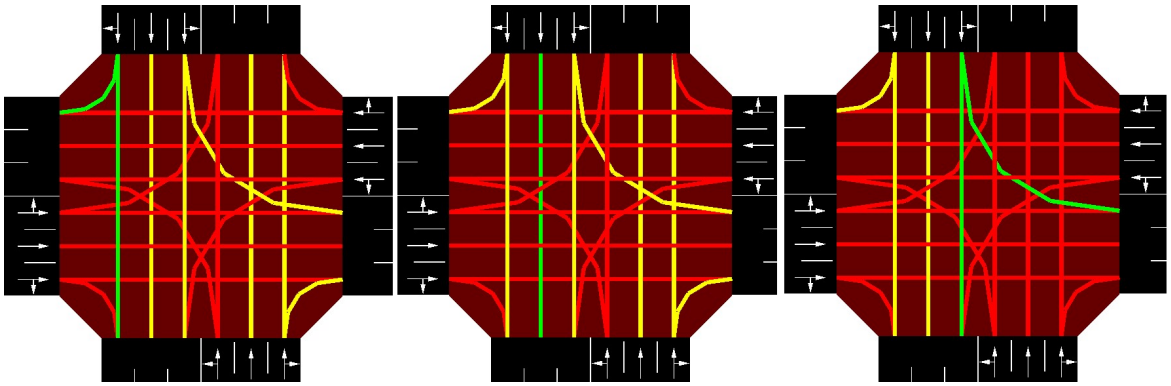
W porównaniu z innymi pracami wykorzystującymi sieci DQN oraz jej rozwinięcia, zdecydowano się na stosunkowo mały rozmiar pojedynczego bufora [50], [74]. Było to spowodowane dużą liczbą agentów. Każdy agent posiadał własną pamięć, dlatego też, gdyby nie ograniczono rozmiaru bufora ich łączny rozmiar byłby znaczny, co w przypadku najbardziej złożonych uruchomień mogłoby wymusić wykorzystanie całej pamięci operacyjnej.

Liczba kroków symulacji między kolejnymi kopiowaniami wag była wyższa niż najczęściej spotykane 10000 kroków, które zostały wykorzystane m.in. w pracy przedstawiającej algorytm DDQN [58]. Było to prawdopodobnie spowodowane mniejszą stabilnością procesu uczenia z powodu większej liczby agentów. Dla mniejszej wartości liczby kroków pomiędzy kopiowaniami (5000) na początku procesu uczenia uzyskiwane wartości funkcji kosztu spadały, jednak po pewnym czasie (około 500 tysięcy kroków) wartości te zaczynały rosnąć. Względną stabilność procesu uczenia uzyskano dla liczby kroków równej 25000, jednak aby zminimalizować zjawisko przesuwającego się celu opisane w części Deep Q-learning zdecydowano się na większą wartość tego parametru.

6.2. Kierunki

Każdy kierunek przejazdu przez skrzyżowanie blokował częściowo inne. W przypadku za długiego trwania pozwolenia na przejazd, prowadziło to do większych opóźnień. W związku z tym, zdecydowano się na premiowanie skoordynowanych sygnałów w ramach jednej grupy. Dodatkową zaletą takiego podejścia było to, że sąsiednie pasy nie kolidowały ze sobą.

Takie podejście ograniczało sekwencyjność kolejności przejazdów, zachęcając do równoczesnego wystawiania zielonego światła przez wielu agentów. Jednocześnie, dopuszczane były indywidualne sygnały, ponieważ nie zawsze konieczna była koordynacja. Dlatego też, w funkcji nagrody wybieranie wspólnych akcji było nagradzane, ale podejmowanie samodzielnych decyzji nie było karane. Motywację dla takiego sposobu operacji przedstawiono na *rysunku 6.1*. Znajduje się na nim skrzyżowanie z oznaczonym wybranym kierunkiem (kolor zielony) oraz kierunkami z nim kolizyjnymi (czerwony). Kolorem żółtym zaznaczono kierunki niekolidujące. Można zauważyć, że najwięcej „zgodnych" kierunków znajduje się w ramach jednej grupy. Ponadto, jeżeli w ramach danego kierunku występuje lewoskręt oraz na przykład jazda prosto, jego aktywacja blokuje prawie całe skrzyżowanie. Było to jedno z większych ograniczeń agenta-pasa.



Rysunek 6.1. Skrzyżowanie z oznaczonymi kolidującymi i niekolidującymi kierunkami

6.3. Ograniczenia

W ramach projektu zastosowano kilka uproszczeń zadania. Przykładowo, podobnie jak w niektórych innych pracach, zdecydowano się na ograniczoną przestrzeń akcji, czyli jedynie na kolor czerwony i zielony. Ciężko byłoby uchwycić istotę światła żółtego i pożądane sposoby jego zastosowania, szczególnie w funkcji nagrody. Zmniejszenie przestrzeni akcji pozwoliło także na większą prędkość procesu uczenia, ponieważ możliwe było ograniczenie liczby kroków symulacji.

Tworzony system był wieloagentowy, dlatego też decyzje podejmowane przez agenty oddziaływały nie tylko na środowisko, ale także na siebie nawzajem. Zależnie od akcji agentów zmieniało się obciążenie ruchem na innych skrzyżowaniach. Najbardziej znamiennym wpływem było blokowanie się agentów nawzajem. W związku z tym duża część kierunków była kolidująca, paradoksalnie procentowo większa w mniej rozbudowanych skrzyżowaniach, konieczne było zapewnienie mechanizmu niepozwalającego agentom na doprowadzenie do przecinania się aktywnych kierunków.

W stworzonym mechanizmie każdy z agentów był odpytywany sekwencyjnie w losowej kolejności o pożądany sygnał. Jeżeli agent decydował o włączeniu zielonego światła i było to możliwe (nie powodowało konfliktu) akcja ta była zrealizowana, a stan skrzyżowania aktualizowany (bez wykonania kroku symulacji, jedynie wewnętrzny stan skrzyżowania). Jeśli akcja nie była możliwa do zastosowania, agent zmuszany był do światła czerwonego i dodawany do listy zablokowanych. Włączenie czerwonego światła było akcją bezkonfliktową, ale mogło odblokować inne agenty, a więc także po tych akcjach była konieczna aktualizacja stanu skrzyżowania. Po rozpatrzeniu akcji wszystkich agentów, ponownie sprawdzane były poprzednio zablokowane agenty - czy zostały odblokowane i czy mogły włączyć zielone światło.

Powyższy mechanizm prawdopodobnie mógł zostać zrealizowany także w ramach funkcji nagrody, jednakże podobnie jak w niektórych pracach, część zasad ruchu lepiej realizowana jest przez proste algorytmy, podczas gdy sam dobór akcji pozostawia się dla agentów ograniczając i upraszczając ich zadanie [75].

6.4. Stan

Każdy stan składał się z określonej liczby obserwacji, czyli danych uzyskiwanych ze środowiska po każdym kroku. W wyniku dokonanej eksploracji hiperparametrów, zdecydowano się na dwanaście kolejnych obserwacji. Miało to na celu zapewnienie dostępu agentom do pozornie nieuchwytnych wartości, jak na przykład dynamiki zmian. Wszystkie pola przekazywane agentom przedstawiono w *tabeli 6.2*. Poza zmiennymi dla których wartości były zbiorami, wszystkie inne pola miały charakter ciągły.

Tabela 6.2. Pola pojedynczej obserwacji, których seria wchodzi w skład stanu

lp.	Wielkość	Zakres wartości
1	Czy mam zielone światło	{0, 1}
2	Czy mam czerwone światło	{0, 1}
3	Czy pętla indukcyjna bliżej skrzyżowania jest włączona	{0, 1}
4	Czy pętla indukcyjna dalej skrzyżowania jest włączona	{0, 1}
5	Zajętość mojego pasa	[0, 1]
6	Czas od ostatniego przełączenia sygnału	[0, 1]
7	Czas od ostatniego czerwonego światła	[0, 1]
8	Czas od ostatniego zielonego światła	[0, 1]
9	Czas oczekiwania pierwszego pojazdu	[0, 1]
10	Względna zajętość mojego pasa	[0, 1]
11	Względny czas oczekiwania pierwszego pojazdu	[0, 1]
12	Zajętość pasa lewego sąsiada	[0, 1]
13	Sygnał lewego sąsiada	{0, 1}
14	Zajętość pasa prawego sąsiada	[0, 1]
15	Sygnał prawego sąsiada	{0, 1}
16	Czas od kiedy wszystkie agenty z grupy mieli światło czerwone	[0, 1]

Wszystkie wartości zostały poddane normalizacji do zakresu [0, 1]. Dla zmiennych czasowych, ich wartości x zostały znormalizowane z ograniczeniem górnym: $y = \min(1, \frac{x}{3600})$. Elementy 1-9 dotyczą wyłącznie agenta, jego sygnałów oraz detektorów. Wartości 10 i 11 opisują stosunek danej wartości agenta względem maksymalnej wartości wśród wszystkich innych agentów i przekazują informacje o wielkości ruchu na innych pasach. Pola 12-15 mówią o danych uzyskiwanych od sąsiednich agentów. W przypadku, gdy sąsiedni agent nie istnieje, te wartości ustawiane są na wartości równe zero (kolejki) oraz światło czerwone (sygnał). Ponieważ funkcja nagrody nie karze za różne sygnały w ramach jednej grupy, takie rozwiązanie zachęca agenty do wspólnego wystawiania światła zielonego. Ostatnia wartość o numerze 16 przekazuje informacje o tym, kiedy ostatnio wszystkie agenty z grupy miały czerwone światło. To pole było konieczne, aby zapobiec nadmiernemu blokowaniu całego skrzyżowania przez jedną grupę.

6.5. Funkcja nagrody

Zaprojektowanie funkcji nagrody w stworzonym wieloagentowym środowisku wymagało znalezienia równowagi między składowymi wchodzącymi w cel tworzonego rozwiązania. Funkcja musiała promować indywidualne zachowania każdego agenta tak, aby w możliwie największym stopniu starał się ograniczyć zajętość swojego pasa. Z drugiej strony, samolubne działania pojedynczego agenta nie mogły doprowadzić do zablokowania pozostałych. Konieczne było również dodanie elementów wspomagających współdziałanie grup w celu minimalizacji czasu, przez który agenty blokowały skrzyżowanie. Funkcja nagrody została stworzona w oparciu o wcześniejsze założenia. Jeżeli agent nie posiada ani lewego, ani prawego sąsiada zostało to uwzględnione jako kolejka o długości zero

z zielonym światłem, przez co nie przełoży się na ewentualną dodatkową nagrodę za skoordynowane działanie.

W *tabeli 6.3* przedstawiono stałe wchodzące w skład funkcji nagrody. Nazwy rozpoczynające się od litery i znaku podkreślenia „K_” oznaczają cząstkowe kary i mają wartości ujemne, te rozpoczynające się od „N_” oznaczają cząstkowe nagrody. Część implementacji funkcji przedstawia *listing 2*. Niektóre wielkości występujące w funkcji nagrody występują bezpośrednio w stanie przedstawionym w *tabeli 6.2*. Wymagającymi objaśnienia pojęciami są:

- *procentZajętości* - pole nr 5,
- *czasOdPrzełączenia* - pole nr 6,
- *największaKolejka* - prawda, jeżeli pole nr 10 jest równe 1.0, czyli, jeżeli agent ma największą kolejkę,
- *czasGrupy* - pole nr 16.

Tabela 6.3. Współczynniki i wartości występujące w funkcji nagrody

lp.	**Wielkość**	**Wartość**	**Objaśnienie**
1	K_ZIELONE_GDY_PUSTY	-5	Blokowanie innych agentów
2	K_ZIELONE_JAK_SĄSIAD_GDY_PUSTY	-2	Zapobiega nadmiernemu inspirowaniu się sygnałem sąsiada
3	K_CZERWONE_GDY_DUŻA_KOLEJKA	-0,7	Agent powinien mieć zielone
4	K_ZBYT_DŁUGI_SYGNAŁ	-2	Blokowanie innych agentów
6	K_ZAJĘTOŚĆ	-2	Skalowana wraz z kolejką
7	N_CZERWONE_GDY_PUSTY	1	Nieblokowanie innych agentów
8	N_ZIELONE_GDY_NIEPUSTY	2	Służy zmniejszaniu kolejki
9	N_ZIELONE_GDY_NAJWIĘKSZA_KOLEJKA	12	Służy priorytetyzacji największej kolejki

Listing 2: Fragment implementacji funkcji nagrody

```
1  if pustyPas && zieloneŚwiatło then
2      if czasOdPrzełączenia >= MINIMALNY_CZAS_FAZY then
3          nagroda = nagroda + K_ZIELONE_GDY_PUSTY
4      if L_SĄSIAD_ZIELONE || P_SĄSIAD_ZIELONE then
5          nagroda = nagroda + K_ZIELONE_JAK_SĄSIAD_GDY_PUSTY
6  else if pustyPas then
7      nagroda = nagroda + N_CZERWONE_GDY_PUSTY
8  else if procentZajętości > 0.2 && zieloneŚwiatło then
9      nagroda =
         nagroda + N_ZIELONE_GDY_NIEPUSTY * procentZajętości
10 else if procentZajętości > 0.5 && ! zieloneŚwiatło then
11     nagroda =
         nagroda + K_CZERWONE_GDY_DUŻA_KOLEJKA * procentZajętości
12 if czasOdPrzełączenia >= ZA_DŁUGI_SYGNAŁ then
13     nagroda = nagroda + czasOdPrzełączenia * K_ZBYT_DŁUGI_SYGNAŁ
14 if największaKolejka && zieloneŚwiatło then
15     nagroda = nagroda + N_ZIELONE_GDY_NAJWIĘKSZA_KOLEJKA
16 if czasGrupy > MINUTA then
17     nagroda = nagroda + czasGrupy * K_CZAS_GRUPY
18 nagroda = nagroda + procentZajętości * K_ZAJĘTOŚĆ
```

6.6. Strategia ϵ-zachłanna

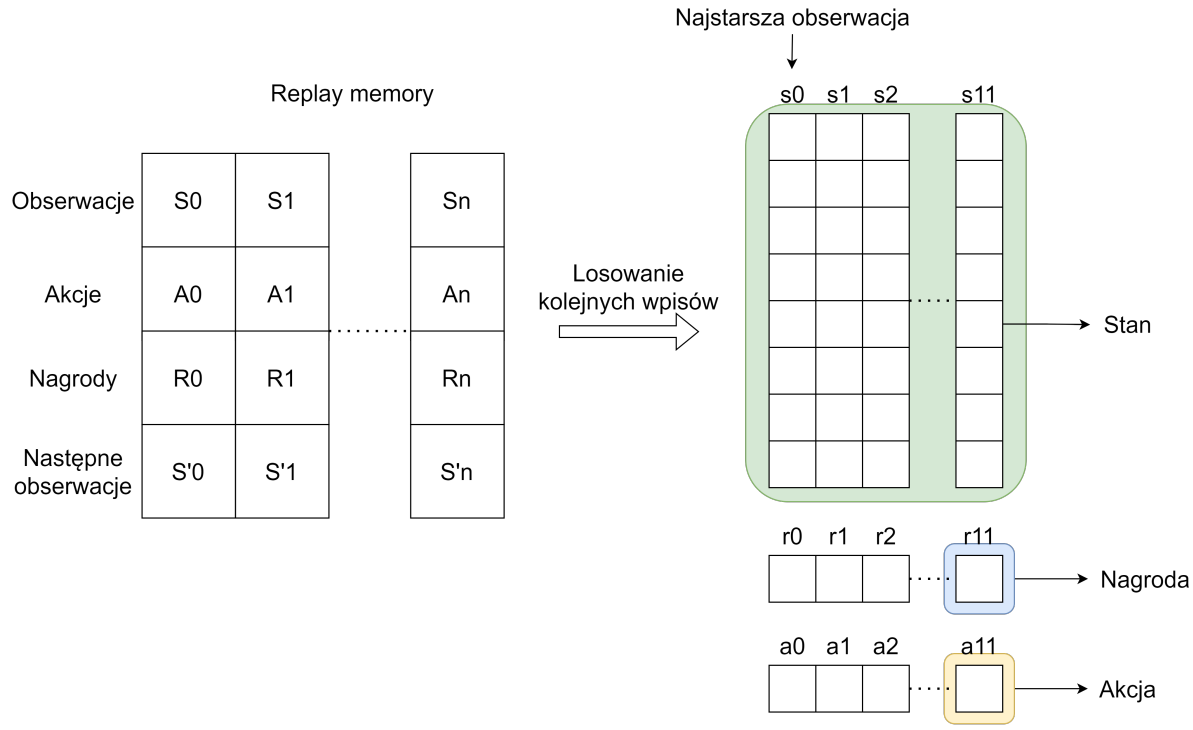
W ramach balansowania eksploracji i eksploatacji, szczególnie w początkowych krokach procesu uczenia, zdecydowano się na wykorzystanie strategii ϵ-zachłannej. Wartość ϵ, która określała prawdopodobieństwo z jakim były podejmowane losowe akcje była liniowo zmniejszana w czasie tak, aby z czasem pozwolić na eksploatację.

Wadą niniejszej strategii było to, że prawdopodobieństwo z jakim wykonana zostanie losowa akcja nie zależy od tego, czego agent zdołał się już nauczyć. W ramach niniejszego projektu nie stanowiło to dużego problemu, ponieważ liczba akcji, które agent mógł wybrać była mała. Dzięki temu, już stosunkowo niewielka (względem innych prac opisujących uczenie ze wzmocnieniem) liczba losowo podjętych decyzji, pozwalała na wstępne przetestowanie i nauczenie się skutków akcji.

6.7. Bufory *experience replay*

Zdecydowano, że każdy z agentów będzie posiadać indywidualny bufor cykliczny, służący jako bufor danych wykorzystywanych w procesie uczenia. Co każdy krok symulacji umieszczano w nim zestaw informacji dotyczących danego agenta: obserwacja, akcja, nagroda, kolejna obserwacja. W każdym kroku uczącym dane wybierane są losowo, od losowych agentów, dopóki nie zostanie zebrana ich wymagana liczba. Uproszczony sche-

mat wyboru danych przedstawiono na *rysunku 6.2*. Część zapisywanych danych była powtórzona, gdy obserwacja z kolejnego wpisu była identyczna z następną obserwacją z aktualnego - $S_n = S'_{n-1}$, jednak dzięki temu losowanie danych do procesu uczenia mogło być zrealizowane w prostszy sposób.



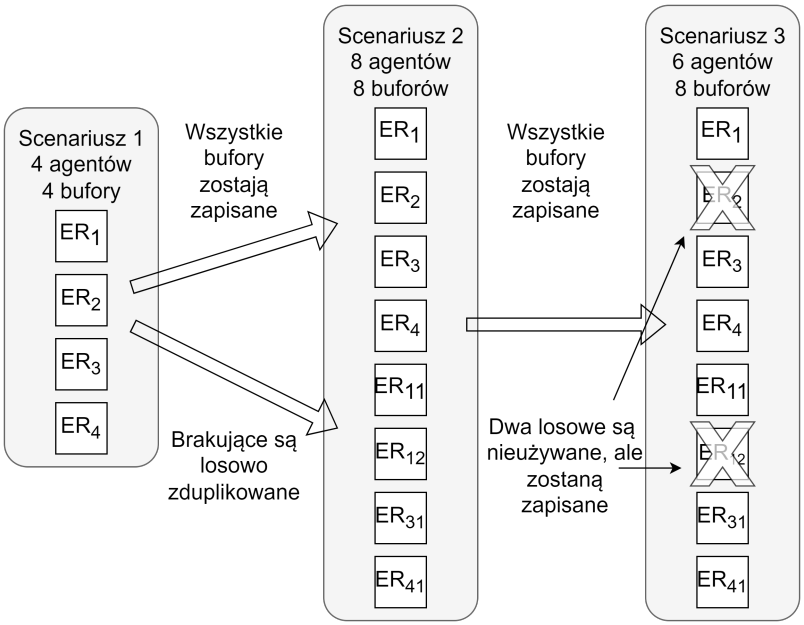
Rysunek 6.2. Schemat losowania danych do procesu uczenia

Agenty na wejściu zawsze otrzymywały stan, w którym przynajmniej przez pięć sekund (minimalny czas trwania fazy) trwała aktualna akcja. W tym celu zmodyfikowano bufor tak, aby do procesu uczenia nigdy nie zwracał stanów pośrednich czyli takich, w których w ciągu najnowszych pięciu obserwacji wystawiana akcja się zmieniała. Taka adaptacja pozwoliła na lepsze dostosowanie agenta do danych wejściowych i zwiększyła jego efektywność, dostarczając wyłącznie dane, których mógł się spodziewać.

Aby ograniczyć wpływ procesu zapominania poprawnego działania agenta (*catastrophic forgetting*), zdecydowano się zastosować ograniczenie w buforze przechowującym obserwacje. Polegało ono na ustaleniu $\frac{1}{10}$ całości każdego bufora jako niemożliwej do nadpisywania. Dzięki temu, możliwe było douczanie sieci na dawnych, nieoptymalnie podjętych decyzjach. Większość trwałych wspomnień przechowywanych w buforach pochodziła z okresu, gdy parametr ϵ nie był jeszcze równy zero.

Zależnie od scenariusza liczba agentów mogła się zmieniać. Zatem, konieczne było stworzenie mechanizmu ich obsługi w momencie, gdy liczba agentów się zwiększała. Z powodu zastosowania predefiniowanego programu *curriculum learning*, w miarę postępowania zaawansowania uczenia, rosło skomplikowanie scenariuszy oraz liczba agentów w nich występujących. Oznaczało to także potrzebę stworzenia mechanizmu przyrostu liczby buforów. Po zakończeniu każdego scenariusza zapisywano wszystkie bufory na dysku.

Rozpoczynając kolejny scenariusz, jeżeli liczba agentów była mniejsza wykorzystywano część z wczytanych buforów, zapisując wszystkie (także te aktualnie niewykorzystywane). Jeśli liczba agentów była większa niż liczba wczytanych buforów, losowo klonowano część z nich tak, aby liczba agentów i buforów była równa. Mechanizm duplikacji powstał w celu zachowania procentowego udziału obserwacji pochodzących z okresu eksploracji. Proces ten przedstawiono na *rysunku 6.3*.



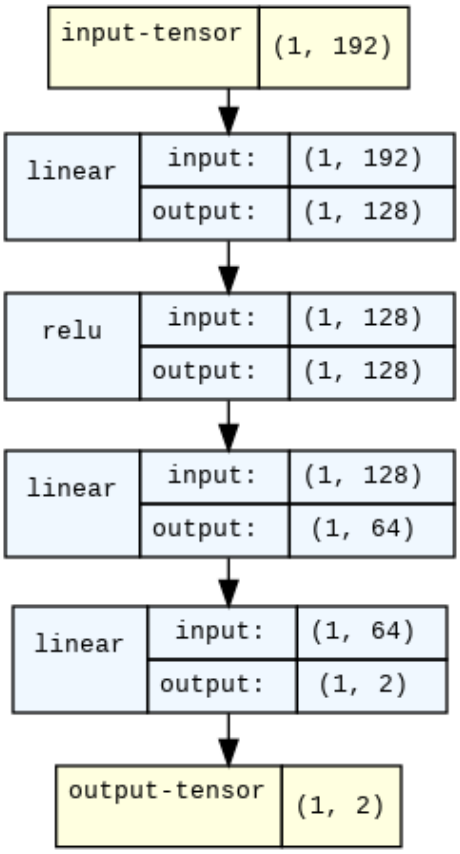
Rysunek 6.3. Mechanizm obsługi buforów *experience replay* podczas procesu uczenia

6.8. Sieć neuronowa

W projekcie po serii eksperymentów, zdecydowano się zastosować trzywarstwową, w pełni połączoną sieć neuronową. Wyłącznie po pierwszej warstwie zastosowano funkcję aktywacji ReLU [76]. Ostateczny kształt zastosowanej sieci został przedstawiony na *rysunku 6.4*. Próbowano zastosować także sieci o innych kształtach i warstwach, jednakże nie pozwoliły one na uzyskanie lepszych wyników.

W literaturze popularne jest stosowanie warstw splotowych, jednakże sprawdzają się one najlepiej podczas przetwarzania obrazów oraz innych danych, w których ważna jest pozycja i hierarchia danych. W kontekście sterowania sygnalizacją świetlną warstwy splotowe są wykorzystywane najczęściej razem ze specjalnym sposobem opisywania aktualnego stanu, nazwanego DTSE [77]. W przypadku niniejszej pracy i obranych założeń nie mógł on zostać zastosowany.

Poza warstwami splotowymi czasem wykorzystywane są także warstwy LSTM [78] [67], których zastosowanie może się okazać szczególnie trafne w częściowo obserwowalnych środowiskach, takich jak to, stworzone w ramach niniejszej pracy. Mimo potencjalnych dobrych wyników, okazało się, że agenty realizujące polityki uzyskane za pomocą sieci LSTM osiągały podobne wyniki jak te, korzystające z wyjściowej sieci, przy jednoczesnym, znacznym wydłużeniu czasu obliczeń.



Rysunek 6.4. Schemat wykorzystanej sieci neuronowej oraz tensorów wejściowych i wyjściowych wraz z wymiarami

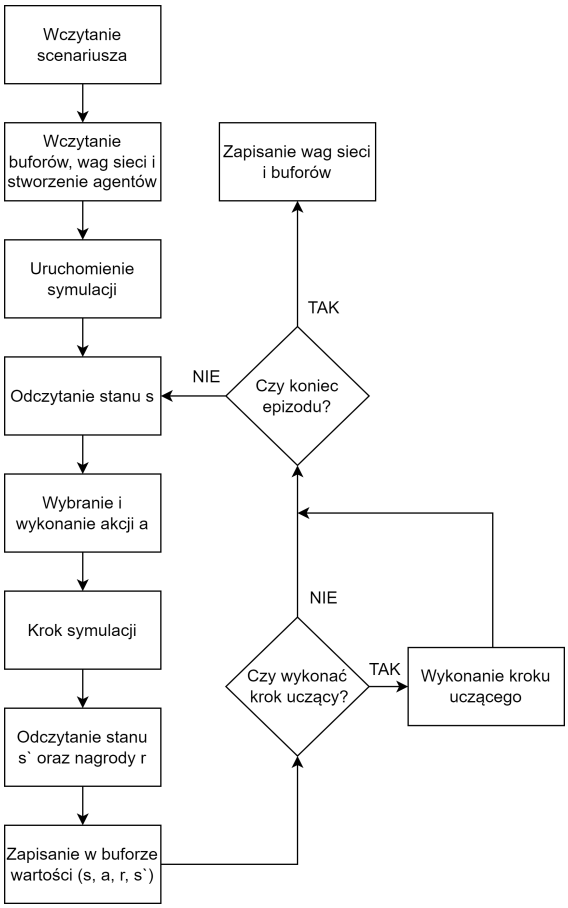
W różnych pracach wykorzystujących MARL, autorzy dostosowują architekturę i liczbę wykorzystywanych sieci neuronowych w zależności od konkretnego problemu. Powiązanie agenta z indywidualną siecią powoduje konieczność stworzenia dodatkowego mechanizmu wymiany informacji. Spotykane jest także zastosowanie wyłącznie jednej, współdzielonej sieci (a właściwie dwóch sieci z powodów opisanych w części Deep Q-learning). Takie podejście zostało wykorzystane w niniejszej pracy. Najistotniejszym atutem takiej metody działania była znaczna oszczędność zasobów sprzętowych wymaganych do procesu uczenia, szczególnie ważna w przypadku skomplikowanych scenariuszy.

Do trenowania sieci wykorzystano optymalizator Adam [79] i funkcję kosztu Hubera (ang. *Huber loss*). Wagi sieci były przekopiowywane z sieci doraźnej do sieci docelowej co 50000 kroków. Aktualizacji wag sieci doraźnej dokonywano co 8 kroków symulacji, za pomocą 32 wylosowanych próbek.

6.9. Przebieg symulacji

Przed wystartowaniem symulacji, uruchamiano skrypty generujące detektory dla wybranej siatki ulic oraz ruch uliczny o jednostajnym natężeniu, z zadanym okresem pojawiania się nowego samochodu. W przypadku kontynuowania rozpoczętego procesu uczenia, wczytywano zapisane bufory *experience replay*, tworzono agenty i uruchamiano symulację.

W każdym kroku symulacyjnym, przed odpytaniem agentów o akcję, zapisywano stan (s) każdego z nich. Następnie dokonywano wyboru akcji (a) w losowej kolejności, w sposób opisany w części 6.3. Ograniczenia. Po wyborze akcji, wykonywano krok środowiska i odnotowywano nagrodę (r) oraz kolejny stan ($s^{'}$). Tak zebrane wartości $(s, a, r, s^{'})$ zapisywano w buforze wspomnień każdego agenta, z którego losowane będą dane wykorzystywane w procesie uczenia. Co określoną liczbę kroków, ze wszystkich buforów losowano próbki uczące, które wykorzystywano w kroku aktualizacji wag sieci neuronowej za pomocą propagacji wstecznej. Następnie, jeżeli od ostatniej aktualizacji wag sieci docelowej upłynęła wystarczająca liczba kroków, dokonywano przekopiowywania wag sieci $\theta^{-} = \theta$. Jeżeli symulacja dobiegała końca, zapisywano wagi obu sieci, numer ostatniego wykonanego kroku oraz stan wszystkich buforów tak, aby możliwe było kontynuowanie uczenia. Natomiast jeżeli symulacja się jeszcze nie kończyła, wracano do początku pętli, to jest do momentu odczytania stanu s. Schemat przebiegu symulacji został przedstawiony na *rysunku 6.5.*



Rysunek 6.5. Schemat przebiegu pojedynczej symulacji

Każda symulacja trwała, dopóki nie został skończony wygenerowany scenariusz, czyli dopóki wszystkie samochody nie zakończyły swoich tras. Ponieważ dla epizodów z dużym obciążeniem ruchu symulacja mogła zostać znacznie wydłużona (zaplanowane samochody pojawiały się później), wprowadzono maksymalny czas trwania symulacji. Możliwe były także sytuacje, gdy agent przez długi czas blokował przejazd samochodów, jednak takie

dane były szczególnie cenne z punktu widzenia procesu uczenia, gdyż dostarczały wiedzę na temat źle podjętych decyzji.

6.10. Proces uczenia

Ponieważ agenty podejmowały decyzje na podstawie określonej liczby obserwacji, każda z nich była umieszczana w oddzielnym buforze FIFO. Pozwalało to na poręczne ich przechowywanie i wykorzystanie do określenia wartości funkcji Q w danym momencie.

Pojedyncze uruchomienie epizodu uczącego trwało bardzo długo. Dlatego też, aby usprawnić całość procesu uczenia stworzono skrypt uczący. Pozwoliło to na automatyczne podążanie za stworzonym scenariuszem, opartym o podejście *curriculum learning*, tj. w miarę postępu procesu nauczania, generowano coraz większy ruch i korzystano z bardziej skomplikowanych siatek ulic (wszystkie przedstawiono w załączniku 1.).

Siatki ulic podzielono na pięć zestawów o różnej, narastającej „trudności", zarówno pod kątem stopnia skomplikowania ruchu (więcej skrzyżowań, pasów, kombinacji kierunków), jak i złożoności obliczeniowej (więcej pasów oznaczało więcej agentów). Generowany z losowym ziarnem ruch uliczny miał równomierne natężenie przez czas trwania całego epizodu. Aby poddać agenty różnorodnym sytuacjom, tworzone przepływy pojazdów charakteryzowały się od przesadnie niskiego do nadmiernie wysokiego natężenia ruchu.

Poza pierwszym etapem procesu uczenia, scenariusz był wybierany losowo z danej grupy. Okres oznacza czas do pojawienia się nowego pojazdu na losowej, krawędziowej drodze dojazdowej. Co każdy epizod w ramach każdego etapu uczenia okres malał o 0,1 s co epizod. Kolejne poziomy procesu wraz z okresem pojawienia się nowego pojazdu są następujące:

- 1 epizod, siatka 4x2, czas 3600s, okres 3 s,
- 10 epizodów, grupa 1., czas 10800s, okres 3 s,
- 10 epizodów, grupa 1., czas 10800s, okres 2,3 s,
- 10 epizodów, grupa 1., czas 10800s, okres 1,9 s,
- 10 epizodów, grupa 2., czas 10800s, okres 3 s,
- 10 epizodów, grupa 2., czas 10800s, okres 2,5 s,
- 10 epizodów, grupa 2., czas 10800s, okres 2,2 s,
- 10 epizodów, grupa 3., czas 10800s, okres 3 s,
- 10 epizodów, grupa 3., czas 10800s, okres 2,4 s,
- 10 epizodów, grupa 3., czas 10800s, okres 2 s,
- 10 epizodów, grupa 4., czas 10800s, okres 3 s,
- 10 epizodów, grupa 4., czas 10800s, okres 2 s,
- 10 epizodów, grupa 4., czas 10800s, okres 1,5 s,
- 15 epizodów, grupa 5., czas 10800s, okres 2,5 s,
- 15 epizodów, grupa 5., czas 10800s, okres 2 s,
- 15 epizodów, grupa 5., czas 10800s, okres 1,6 s.

Z powodu zastosowanego, stosunkowo dużego maksymalnego czasu symulacji, niniejsze podejście dopuszcza większą „bezwładność" procesu uczenia. Sumując czasy trwania

poszczególnych epizodów, minimalna liczba kroków symulacji to 1785600. W rzeczywistości, liczba ta zazwyczaj przekraczała 2 miliony kroków. W innych pracach poruszających zagadnienie wykorzystania uczenia ze wzmocnieniem do sterowania sygnalizacji świetlnych autorzy narzucają bardziej restrykcyjne ograniczenia czasu trwania każdego epizodu, jednakże korzystali oni z prostszych siatek ulic.

Zastosowanie wielu agentów w ramach pojedynczego skrzyżowania wymusiło również bardziej dokładne zbieranie obserwacji, także tych negatywnych.

7. Testowanie i weryfikacja rozwiązania

Ten rozdział zawiera opis testów przeprowadzonych na stworzonym rozwiązaniu oraz analizę uzyskanych wyników. Przedstawiono rezultaty i sformułowane wnioski na podstawie przeprowadzonego procesu uczenia. Omówiony został sposób weryfikacji zaimplementowanego systemu oraz jego zgodności z opisanymi wcześniej założeniami i celami. Opisano także przebieg i zaprezentowano rezultaty ewaluacji projektu.

7.1. Sprzęt i wydajność

Uczenie i testowanie rozwiązania wymagało stosunkowo mocnego komputera osobistego, zasobnego szczególnie w pamięć RAM. Ponieważ cały proces trwał bardzo długo, w celu skrócenia potrzebnego czasu niektóre z etapów uruchomiono na kilku urządzeniach pomocniczych. Główny komputer, na którym była przeprowadzona większość obliczeń posiadał następujące podzespoły:

- procesor Intel Core i7-7700HQ,
- 32 GB pamięci RAM o taktowaniu 2666 MHz,
- kartę graficzną Nvidia GeForce GTX 1050.

W pozostałych urządzeniach znajdowały się komponenty o podobnej wydajności, jednak posiadały o połowę mniej pamięci RAM. Nie odnotowano znaczącej poprawy działania procesu między urządzeniami korzystającymi z szybkich dysków SSD w porównaniu do tych używających wyłącznie dysków HDD.

Pojedyncze uruchomienie skryptu uczącego na głównym komputerze trwało około pięciu dni. Najwięcej czasu zajmowały operacje na pamięci - przetwarzanie danych uzyskanych z sieci i symulatora oraz zapisywanie do buforów. Innymi działaniami, które wymagały znacznego nakładu obliczeniowego była komunikacja z symulatorem. Był on uruchamiany w oddzielnym procesie. Dużo operacji kosztowało również korzystanie z i aktualizowanie sieci neuronowej. Zastosowana sieć nie była skomplikowana, dlatego też użycie karty graficznej z CUDA nie usprawniło znacznie całego procesu. Z powodu wybranej architektury rozwiązania większość operacji musiała być wykonywana sekwencyjnie, a więc największy wpływ na czas trwania obliczeń miało taktowanie procesora i rozmiar jego pamięci podręcznej oraz rozmiar pamięci RAM. Nie było możliwe także uruchomienie wielu procesów uczących równolegle, gdyż bardzo szybko zabrakłoby pamięci operacyjnej.

Jak pokazały narzędzia monitorujące zużycie zasobów podczas uruchomień na wielu komputerach, 16 GB pamięci RAM nie było wystarczające do sprawnego przeprowadzenia całego procesu uczenia. Szczególnie czasochłonne były późniejsze, bardziej skomplikowane epizody uczące. Uruchomienia na tych maszynach potrafiły zająć co najmniej dwukrotnie więcej czasu. Prawdopodobnie był to efekt czasochłonnych operacji na partycji wymiany.

Obserwowane, znaczne zapotrzebowanie na pamięć było skutkiem wybranego podejścia agenta-pasa. Każdy z agentów posiadał własny bufor *experience replay*. Ponieważ liczba agentów rosła w miarę zaawansowania procesu uczenia od około 8 do prawie 150, wraz ze wzrostem ich liczby, konieczne było wykorzystanie większej pamięci.

7.2. Wyniki procesu uczenia

Jako rezultat procesu uczenia uzyskano sześć modeli sieci neuronowej. Budową nie różniły się między sobą. Ich liczba wynikała z chęci dokładnego sprawdzenia stworzonego rozwiązania oraz wpływu losowości na końcowe rezultaty. Z powodu decyzji podjętych w fazie projektowej, całkowita liczba kroków trwania uczenia była zróżnicowana od prawie 2 do 2,5 miliona, co odpowiadało 24-29 dniom w symulatorze. Początkowe epizody uczące zawsze kończyły się w zaplanowanym czasie, jednak te z grupy 4 i 5 często musiały być przerywane z powodu zbyt długiego czasu trwania, co spowodowane było znacznym natężeniem ruchu. Przybliżone liczby kroków wykonanych podczas uczenia modeli zaprezentowano w *tabeli 7.1*.

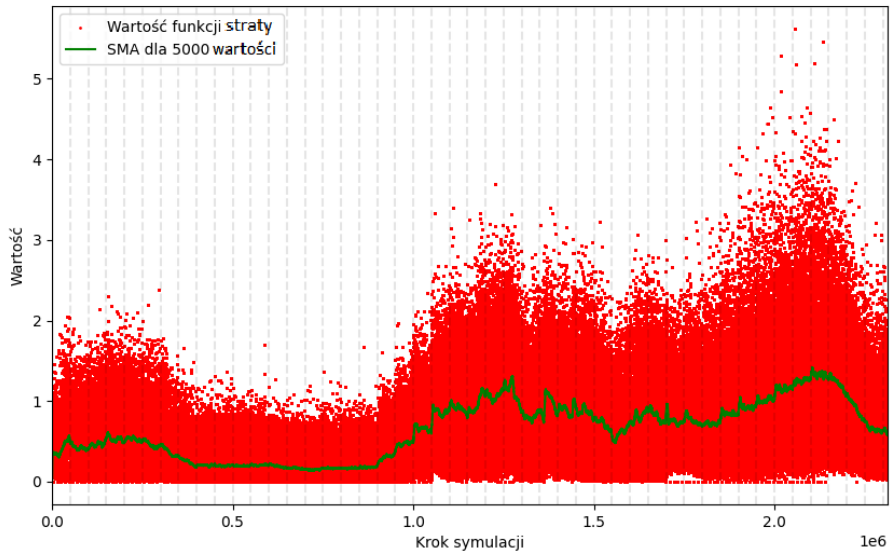
Tabela 7.1. Nauczone modele sieci neuronowej wraz z przybliżonymi łącznymi liczbami kroków symulacji

Nazwa	Liczba kroków
m1	2,4 mln
m2	2,3 mln
m3	1,8 mln
m4	2,0 mln
m5	1,9 mln
m6	2,2 mln

Na *rysunku 7.1* przedstawiono uzyskane wartości funkcji straty dla jednego z modeli, uczonego przez prawie 2,5 miliona kroków. Kolorem czerwonym przedstawiono kolejne wartości otrzymane w ramach aktualizacji wag sieci doraźnej. W celu lepszego uchwycenia trendu zmian uzyskiwanych wartości, kolorem zielonym zaznaczono prostą średnią kroczącą (SMA) dla ostatnich 5000 odczytów. Wartość tę dobrano tak, aby z jednej strony wystarczająco wygładzała zaszumione dane, a z drugiej nie wprowadzała opóźnienia utrudniającego analizę wykresu. Pionowymi przerywanymi liniami zaznaczono momenty aktualizacji wag sieci docelowej.

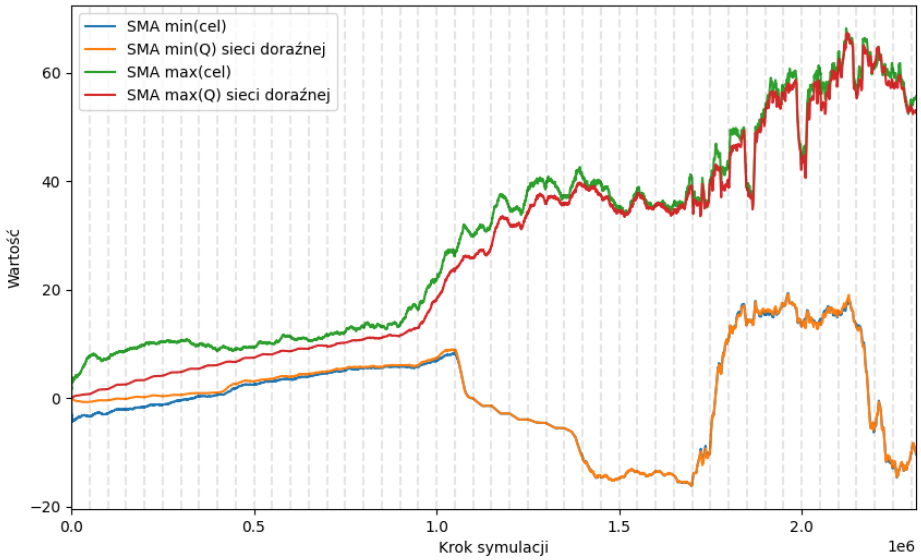
Jak można zauważyć, otrzymywane wartości funkcji straty znacząco rosną po milionowym kroku. Najprawdopodobniej jest to spowodowane wzrostem trudności aktualnie wykorzystywanych epizodów w procesie uczenia. Dla kroków poniżej 1 miliona stosowane były proste siatki ulic, składające się wyłącznie z jednego skrzyżowania. Powyżej tego kroku wybrano siatki składające się z trzech skrzyżowań. Przyczyną wzrostu tych wartości w okolicach dwumilionowego kroku też mogła być powiązana ze zmianą scenariuszy. W okolicach tego kroku zaczynały być wykorzystywane rozbudowane siatki z 9 skrzyżowaniami, jednakże dla początkowych epizodów w ramach tej grupy natężenie było małe. Dodatkowo, poprzez zmianę na większe siatki stosunkowa liczba samochodów na skrzyżowanie była jeszcze mniejsza. Natężenie ruchu powoli rosło, aż osiągało maksymalne wartości pod koniec cyklu uczenia.

Na *rysunku 7.2* przedstawiono SMA dla ostatnich 25000 wartości uzyskiwanych podczas aktualizacji wag. Były to maksymalne i minimalne wartości (na wykresie SMA min i max(Q)) obliczone dla każdej próbki danych przez sieć doraźną. Na rysunku przedstawiono również



Rysunek 7.1. Uzyskane wartości funkcji straty podczas jednego z procesów uczenia

analogicznie maksymalne i minimalne wartości funkcji celu wyznaczanych za pomocą *równania 3.5* (na wykresie SMA min i max(cel)), a więc te wartości, na podstawie których otrzymywano wynik funkcji straty.



Rysunek 7.2. Przebieg przybliżonych wartości funkcji Q uzyskiwanych z sieci doraźnej i docelowej podczas jednego z procesów uczenia

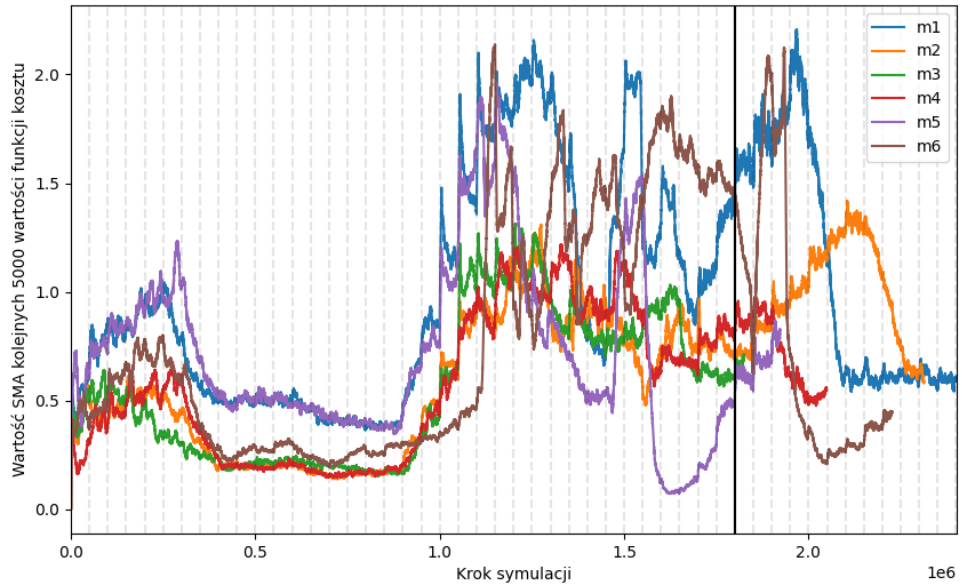
Wartości otrzymywane za pomocą sieci doraźnej podążają za tymi otrzymywanymi z użyciem sieci docelowej. Było to pożądane zjawisko, oznaczające, że uczenie sieci działa poprawnie. Nagłe obniżenia wartości maksymalnych i minimalnych średnich nakładają się w czasie z momentami, gdy wartość funkcji straty rosła. Jest to powiązane zjawisko, ponieważ wraz z napotkaniem nowych obserwacji, model nie umiał poprawnie ich obsłużyć, w wyniku czego otrzymywano bardzo niskie lub wysokie (zależnie od momentu) nagrody.

Minimalne wartości funkcji *Q* dla okolic pierwszego wzrostu funkcji straty (około 1,5 mln krok) sugerują, że był to pierwszy okres, gdy agenty napotkały na prawdziwie trudne do obsłużenia scenariusze, jak na przykład natężenie ruchu uniemożliwiające płynne przejazdy pojazdów. Występowało to dodatkowo z niespotykaną wcześniej sytuacją, gdy skrzyżowań było więcej niż jedno. Wartości maksymalne ulegały zmianom, ponieważ najprawdopodobniej sieć nie była w tym momencie jeszcze do końca nauczona prawdziwej wartości funkcji *Q* dla stanów z małymi kolejkami samochodów. Wartości minimalne maleją, gdyż uzyskano wtedy obserwacje zawierające długie kolejki, albo oznaczające niewystarczający stopień kooperacji między agentami, co przełożyło się na bardzo niską nagrodę. Okolice dwumilionowego kroku to najbardziej rozbudowane siatki ulic o najmniejszym natężeniu ruchu. Świadczą o tym rosnące i dodatnie wartości średnich obu minimów. Ostatnie kroki skryptu uczącego to ponownie nadmierny ruch uliczny dla najbardziej rozbudowanych siatek ulic powodujący spadek spodziewanych wartości przyszłych nagród.

Dane zbierane podczas procesu uczenia były mocno zaszumione z powodu losowania w krokach uczących najpierw agentów, a następnie konkretnych próbek, które miały wchodzić w skład zestawu danych. W przypadku, gdy wylosowane dane były tymi zapisanymi na stałe, najprawdopodobniej sieć została już na nich wielokrotnie uczona i otrzymywana wartość funkcji straty była niska. Mogło się zdarzyć też, że został wylosowany stosunkowo nowy, niespotykany wcześniej zestaw danych - w takim przypadku wartość funkcji osiągała większe wartości. Aby przedstawić zebrane dane w możliwie najczytelniejszy sposób oraz wyszczególnić trendy występujące w procesie uczenia, zdecydowano się na skorzystanie z SMA zamiast prezentować wszystkie uzyskane wyniki.

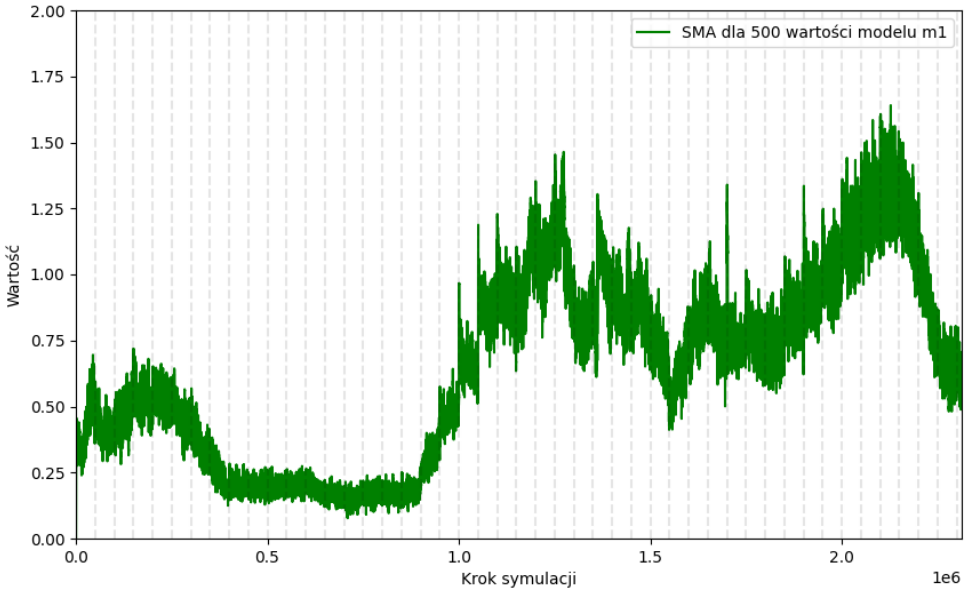
Ważną obserwacją jest także fakt, że wykorzystanie bufora obserwacji powodowało opóźnienie między zapisaniem a wykorzystaniem stanu do aktualizacji wag sieci. Było ono tym większe, im większy był rozmiar bufora. Było to związane z prawdopodobieństwem wylosowania względnie nowych stanów, wobec liczby wszystkich zapisanych danych. W związku z tym, ograniczenie rozmiaru pamięci agentów spowodować mogło szybsze reagowanie na zmiany występujące w środowisku. Z drugiej strony, mały bufor mógł przełożyć się na szybsze zapominanie wiedzy uzyskanej na podstawie już wymazanych obserwacji.

Na *rysunku 7.3* przedstawiono przybliżone zmiany uzyskiwanych wartości funkcji straty dla każdego z modeli. Ponownie zastosowano SMA dla 5000 kroków w celu zwiększenia czytelności wykresu. Pionowa czarna linia dla kroku 1800000 wskazuje przybliżony moment, w którym proces uczenia powinien się skończyć. Ewentualne przekroczenia tego kroku były spowodowane dopuszczonym przedłużeniem epizodów uczących. Najbliżej wspomnianego kroku proces uczenia zakończył model *m3*. Znaczne różnice w wartościach funkcji straty dla różnych modeli wynikają najpewniej z losowości. Nie stworzono dokładnego predefiniowanego programu uczącego, jedynie przybliżony plan dopuszczający ograniczoną przypadkowość. Ponadto generowany ruch korzystał z losowego ziarna.



Rysunek 7.3. Uzyskane wartości funkcji straty dla wszystkich modeli

Pomimo wspomnianej przypadkowości, można zauważyć kilka okresów wspólnych dla każdego z modeli. Pierwsze około ćwierć miliona kroków to proste przykłady, podczas których losowo zainicjalizowane sieci uczone są poprawnego działania w środowisku. Okres 250000 - 1000000 może sugerować, że nastąpiło przeuczenie sieci, albo też kolejne wygenerowane epizody były bardzo do siebie podobne. Pozwala to wysunąć hipotezę, że można byłoby skrócić ten etap uczenia, albo też wprowadzić większe zróżnicowanie epizodów. Od okolic milionowego kroku dla niektórych sieci następuje okres bardzo dużej zmienności wartości, chociaż modele *m2, m3* i *m4* doświadczają jej w mniejszym stopniu.



Rysunek 7.4. Uzyskane wartości funkcji straty dla modelu m1

Charakterystyczne piki średniej, szczególnie widoczne dla modelu *m1* po 1000000 kroku

wynikają z aktualizacji wag sieci docelowej. Zwracane wartości funkcji straty znacznie wtedy rosną, ponieważ przesunięciu ulegają cele, wobec których uczona jest sieć doraźna. Dla wyszczególnienia tego zjawiska, na *rysunku 7.4* przedstawiono przebieg SMA dla 500 wartości modelu *m1*. Także na *rysunku 7.1* można zaobserwować występowanie nagłych wzrostów wartości.

7.3. Skrypt testowy

W celu sprawdzenia jakości stworzonego rozwiązania, każdy z wytrenowanych modeli uruchomiono na niektórych ze stworzonych siatek ulic. W większości przypadków były to już wcześniej wykorzystywane w etapie uczenia układy drogowe. Siatka „random" składała się z dróg o 1- pasach i 8 skrzyżowaniach kontrolowanych przez sygnalizację świetlne. Została wygenerowana wyłącznie w celu testowania. Nie była ona użyta na etapie uczenia.

Podczas generowania natężenia ruchu w etapie weryfikacji za każdym razem wykorzystywano identyczne, losowo wybrane ziarno. Każdy z epizodów testowych trwał domyślnie 3 godziny, czyli 10800 kroków. Podobnie jak dla procesu uczenia, dopuszczono przedłużenie maksymalnego czasu testu, jednak zmniejszono go do 6 godzin. Po ukończeniu wszystkich przejazdów zaplanowanych w ciągu podstawowego czasu, lub po upłynięciu maksymalnego czasu trwania symulacji przechodzono do następnego zaplanowanego scenariusza. W celach porównawczych, na stworzonym skrypcie testowym uruchomiono także sterowanie akomodacyjne [39] o domyślnym programie zaimplementowanym w symulatorze SUMO. Aby akcje, które oba kontrolery mogły wykonywać były takie same, wyłączono w nim możliwość włączania żółtego światła i zmniejszono minimalny czas trwania każdego sygnału (do 5 sekund, analogicznie jak dla agentów), co przełożyło się na większą elastyczność algorytmu poprzez pozwolenie na szybszą zmianę świateł.

- lp. 1, siatka „T", okres 3 s
- lp. 2, siatka „T", okres 2 s
- lp. 3, siatka „T", okres 1 s
- lp. 4, siatka „4x2", okres 3 s
- lp. 5, siatka „4x2", okres 2 s
- lp. 6, siatka „4x2", okres 1 s
- lp. 7, siatka „4x3 - wyłączne lewoskręty", okres 2,8 s
- lp. 8, siatka „4x3 - wyłączne lewoskręty", okres 1,8 s
- lp. 9, siatka „4x3 - wyłączne lewoskręty", okres 0,8 s
- lp. 10, siatka „4x4" - okres 2,7 s
- lp. 11, siatka „4x4" - okres 1,7 s
- lp. 12, siatka „4x4" - okres 0,7 s
- lp. 13, siatka „siatka3" - okres 2,6 s
- lp. 14, siatka „siatka3" - okres 1,6 s
- lp. 15, siatka „siatka3" - okres 0,6 s
- lp. 16, siatka „vargrid" - okres 2,5 s
- lp. 17, siatka „random" - okres 1,4 s
- lp. 18, siatka „random" - okres 0,4 s

Malejące okresy pojawiania się nowych samochodów w ramach każdej siatki wynikały z chęci przetestowania każdego układu pod różnymi natężeniami ruchu. Wraz ze wzrostem skomplikowania siatek ulic, wzrasta liczba samochodów, które mogą znaleźć się w danym scenariuszu oraz natężenie ruchu potrzebne, aby uzyskać efekt zakorkowania.

W każdym ze scenariuszy zebrano dane dotyczące ruchu wszystkich pojazdów zawierających indywidualną stratę czasu każdego z samochodów. Zmierzono także rozmiary kolejek pojazdów na każdym z pasów dojazdowych w każdym kroku symulacji.

7.4. Proces testowania

W odróżnieniu od fazy uczenia, proces testowania przebiegał o wiele szybciej. Wymagał także o wiele mniej pamięci operacyjnej. Dzięki temu możliwe było równoczesne uruchomienie skryptu testowego dla wielu modeli. Pod względem zużycia zasobów, najbardziej wykorzystywany był procesor. Ciągłe zapisywanie obserwacji do buforów nie było konieczne, dlatego też możliwe było uruchomienie czterech równoczesnych instancji testów. Łączny czas trwania tego procesu wynosił około 3 dni. Mimo zmniejszenia zapotrzebowania na pamięć RAM, ostatnie uruchomienia zajmowały większość całego procesu.

7.4.1. Wyniki przeprowadzonych testów

Wszystkie zmierzone i wyznaczone wielkości przedstawiono w tabelach umieszczonych w załączniku 2.). W *tabeli 7.2* przedstawiono porównanie obliczonych danych statystycznych dla algorytmu akomodacyjnego oraz najlepszego dla danego rozwiązania agenta-pasa. Pogrubioną czcionką wyróżniono lepszą z uzyskanych wartości. O ile było to możliwe, wyróżniono rozwiązanie, które sprawdziło się lepiej. Skrótem scz. określono stratę czasu, ak. oznacza algorytm akomodacyjny, służący do porównania. Liczba tras oznacza liczbę wykonanych lub zaplanowanych przejazdów samochodów.

Na podstawie tabeli można wyróżnić niektóre scenariusze testowe. W większości przypadków wartości uzyskiwane przez algorytm i modele były przybliżone lub miały zbliżone rzędy wielkości. W niektórych uruchomieniach, na przykład epizodach 3, 5, 6, 12, 15 nauczone modele potrafiły w całkowicie różny sposób sterować ruchem. Najlepsze z modeli osiągały wówczas kilkukrotnie niższe straty czasu względem najgorszych. Mimo, iż modele w niektórych epizodach uzyskiwały lepsze średnie opóźnienia, uznano je za gorsze względem algorytmu akomodacyjnego. Sytuacja taka miała miejsce na przykład dla 18, ostatniego scenariusza testowego. Było to spowodowane znacznie mniejszą liczbą samochodów, które ukończyły wygenerowane trasy. Najprawdopodobniej oznaczało to, że jeden z agentów blokował część skrzyżowania przez długi czas, co uniemożliwiało rozładowanie poszczególnych kolejek.

Tabela 7.2. Wyniki algorytmu akomodacyjnego oraz najlepszego modelu dla wszystkich siatek

lp. (Liczba tras)	**Algorytm / model**	**Liczba ukończonych tras**	**Σ scz. [s]**	$\overline{\textbf{scz.}}$ **[s]**	$\sigma_{\overline{\textbf{scz.}}}$ **[s]**	**Med. scz. [s]**
1 (2442)	**ak.**	2442	**1007**	**0,41**	**1,28**	0,00
	m6	2442	1359	0,55	1,75	0,00
2 (3619)	**ak.**	3619	**3368**	**0,93**	**2,55**	0,00
	m4	3619	4954	1,37	2,83	0,00
3 (7225)	ak.	7225	91906	12,72	13,93	8,00
	m2	7225	**33798**	**4,68**	**4,30**	**4,00**
4 (2671)	ak.	2671	4481	1,68	5,26	0,00
	m6	2671	**1141**	**0,43**	**1,83**	0,00
5 (4027)	ak.	4027	18155	4,51	9,36	0,00
	m6	4027	**6006**	**1,49**	**4,02**	0,00
6 (8081)	ak.	8081	897939	111,12	103,16	93,00
	m5	8081	**84577**	**10,47**	**12,79**	**6,00**
7 (2857)	ak.	2857	12211	4,27	4,70	3,00
	m6	2857	**1355**	**0,47**	**1,92**	0,00
8 (4464)	ak.	4464	21172	4,74	4,94	3,00
	m4	4464	**7476**	**1,67**	**4,06**	0,00
9 (10056)	ak.	10056	178331	17,73	18,04	12,00
	m5	10056	**115085**	**11,44**	**13,90**	**7,00**
10 (2972)	ak.	2972	15594	5,25	6,44	2,50
	m6	2972	**5894**	**1,98**	**5,75**	**0,00**
11 (4735)	ak.	4735	30684	6,48	**8,20**	3,00
	m6	4735	**24202**	**5,11**	9,92	**0,00**
12 (11583)	**ak.**	11583	**363561**	**31,39**	68,06	**16,00**
	m5	11583	378011	32,63	**33,38**	23,00
13 (3465)	ak.	3465	40298	11,63	12,38	8,00
	m6	3465	**11034**	**3,18**	**6,50**	**0,00**
14 (5615)	ak.	5615	85711	15,26	15,62	11,00
	m6	5615	**40568**	**7,22**	**10,42**	**3,00**
15 (18000)	**ak.**	**14945**	9710272	**649,73**	**1968,42**	63,00
	m2	10728	6491613	605,11	972,10	305,00
16 (4320)	ak.	4320	110274	25,53	18,46	23,00
	m6	**4320**	**11866**	**2,75**	**5,98**	**0,00**
17 (7715)	**ak.**	7715	**85762**	**11,12**	**13,26**	**7,00**
	m4	7715	167867	21,76	21,02	17,00
18 (27000)	**ak.**	**23987**	4968972	207,15	268,77	116,00
	m1	19019	**2404338**	**126,42**	**125,86**	**90,00**

7.4.2. Wyniki poszczególnych kontrolerów

W *tabeli 7.3* umieszczono przetworzone wyniki pod kątem, ile razy każde z rozwiązań charakteryzowało się najlepszym lub najgorszym sterowaniem ruchu. Modele *m1* oraz *m3* osiągały zdecydowanie najgorsze wyniki, natomiast najlepiej radził sobie *m4*. Światła akomodacyjne, wobec których porównywane były modele, kilkukrotnie były zarówno najlepsze, jak i najgorsze.

Tabela 7.3. Nauczone modele sieci neuronowej wraz z przybliżonymi łącznymi liczbami kroków symulacji

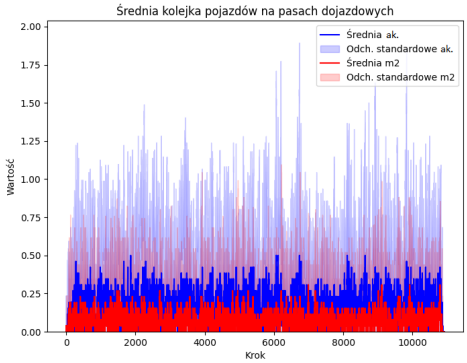
Nazwa	Ile razy najlepszy	Ile razy najgorszy
światła akomodacyjne	6	7
m1	0	3
m2	1	0
m3	0	6
m4	7	0
m5	2	2
m6	2	0
suma wyników modeli	12	11

Chociaż algorytm akomodacyjny najwięcej razy osiągnął najgorszy wynik, z powodu nadreprezentacji modeli względem świateł akomodacyjnych stosowne wydaje się grupowe porównanie obu rozwiązań. W takim przypadku, rezultaty obu grup nie różnią się znacznie. Interesującym faktem jest to, że najgorsze modele należały do ekstremalnych pod względem czasu uczenia: *m1* uczył się najdłużej, a *m3* najkrócej. Może to świadczyć o przeuczeniu modelu *m1*. Inną możliwością był szczególny wpływ losowanych wartości. Przykładowo, niewykluczone, że podczas procesu uczenia *m3* do aktualizacji wag stosunkowo częściej wybierane były nowsze obserwacje, pozwoliło to na lepsze dopasowanie się do aktualnie napotykanych sytuacji i w konsekwencji obserwowane niższe wartości funkcji strat, co przełożyło się na krótszy łączny czas trwania procesu uczenia. Zupełnie przeciwny proces mógł zadziałać podczas uczenia *m1*.

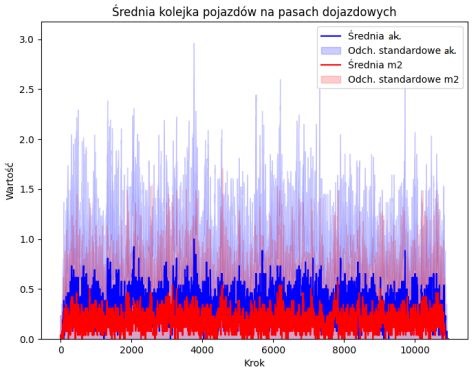
7.4.3. Kolejki samochodów

Ważniejszymi pod kątem weryfikacji spełnienia założeń przez stworzone rozwiązanie były testy o numerach 13-18. Było to spowodowane znacznym zróżnicowaniem skrzyżowań występujących w siatce ulic. Szczególnie ważne były uruchomienia 17 i 18, ponieważ ta siatka ulic nie była wcześniej wykorzystywana. Dla „siatki3”, zarówno w przypadku małego, jak i dużego natężenia ruchu wszystkie modele sprawdziły się o wiele lepiej od algorytmu akomodacyjnego. Na *rysunkach 7.5* oraz *7.6* przedstawiono, jak wyglądały średnie kolejki wraz z wyznaczonymi odchyleniami standardowymi dla modelu *m2* oraz świateł akomodacyjnych. W obu przypadkach średnia liczba oczekujących samochodów jest w miarę stabilna, jednakże dla obu odchylenie standardowe jest znaczne.

Do porównania został wybrany *m2*, ponieważ poradził sobie najlepiej spośród modeli dla dużego natężenia ruchu. Nie osiągnął on najmniejszej średniej straty spośród wszyst-

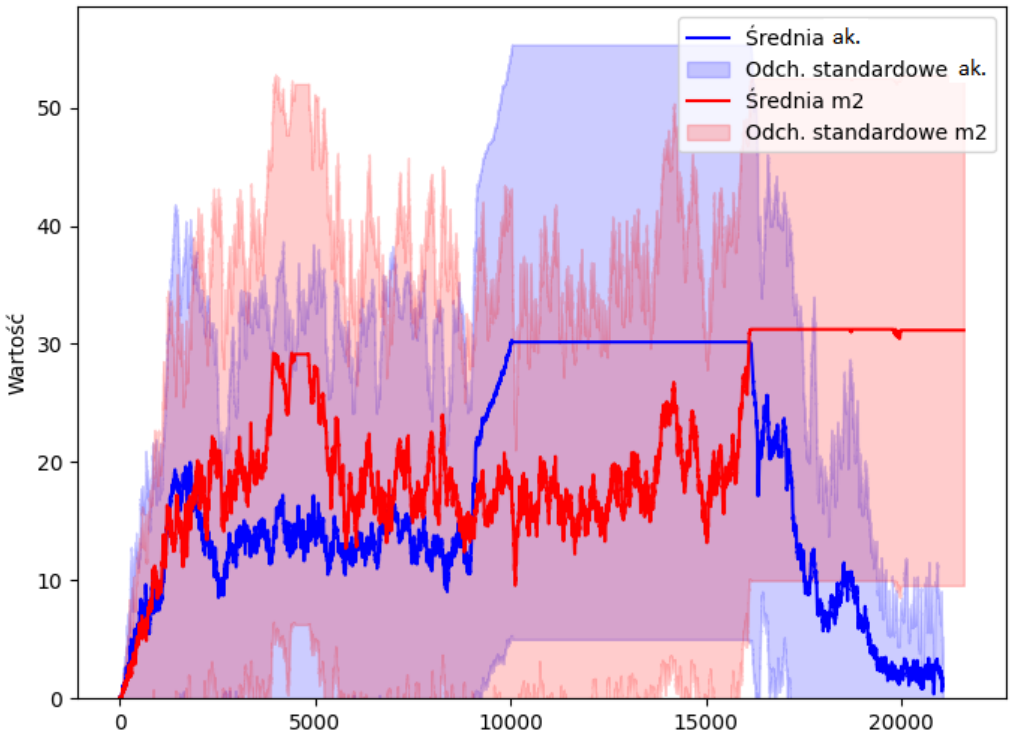


Rysunek 7.5. Średnie kolejki samochodów na pasach dojazdowych dla małego obciążenia ruchem



Rysunek 7.6. Średnie kolejki samochodów na pasach dojazdowych dla średniego obciążenia ruchem

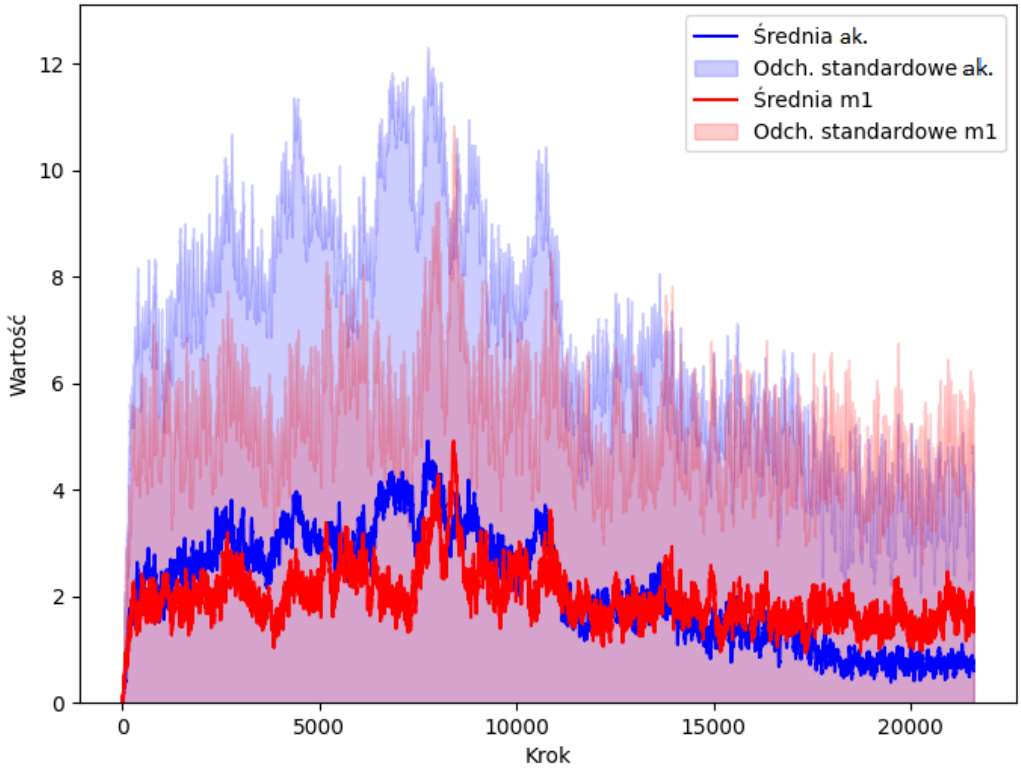
kich modeli, jednakże najwięcej samochodów ukończyło zaplanowaną trasę. Ze wszystkich wygenerowanych 14945 przejazdów, test z *ak.* został zakończony przez 12570 samochodów. Zastosowanie agenta-pasa z *m1* pozwoliło na ukończenie 10728 tras. Najgorszy z badanych, *m6* osiągnął liczbę jedynie 567 przejazdów. Średnie kolejki na pasach dojazdowych przedstawiono na *rysunku 7.7*. Na jego podstawie można zauważyć, że uruchomiony ruch był blokujący dla niektórych kierunków, także przy zastosowaniu sygnalizacji akomodacyjnej. Objawia się to nagłym wzrostem średniej wartości. Niewykluczone również, że jeden z kierunków zamiast dopuścić inne do przejazdu, wystawiał stale zielone światło.



Rysunek 7.7. Średnie kolejki samochodów na pasach dojazdowych dla dużego obciążenia ruchem

Dla siatki *random* o dużym natężeniu ruchu, wyniki średnich strat czasu wyglądają podobnie dla większości modeli. Osiągnięta przez *m2* maksymalna kolejka w przypadku

siatki „siatka3” z dużym ruchem liczyła 25 pojazdów. Scenariusz wykorzystujący układ „random” o dużym natężeniu ruchu, w którym zastosowano *m1* spowodował powstanie maksymalnej kolumny 56 pojazdów. Mimo tego, średnie wartości kolejek osiągane w pierwszym wspomnianym scenariuszu były od kilku do kilkunastokrotnie razy większe względem drugiego przypadku. Zmiany średnich kolejek dla dwóch najlepszych kontrolerów przedstawiono na *rysunku 7.8*.



Rysunek 7.8. Średnie kolejki samochodów na pasach dojazdowych dla dużego obciążenia ruchem

Rozwiązanie oparte na agentach sprawdzało się najlepiej w epizodach o niskim do umiarkowanego natężeniu ruchu. Najprawdopodobniej ma to związek z wybraną metodą komunikacji agentów. Jeżeli liczba pojazdów nie była nadmierna, agenty ze sobą współpracowały, dopuszczając inne operacje i nie blokując pozostałych agentów. Wygląda na to, że mocną stroną zaprojektowanego podejścia była możliwość dynamicznego tworzenia grup sygnałowych, w zależności od potrzeby. Sterowanie akomodacyjne nie pozwalało na taką możliwość, wykorzystując predefinowany zestaw faz świateł. W przypadku znacznego natężenia ruchu chęć ograniczenia indywidualnych kolejek każdego z agentów była większa, niż motywacja do skoordynowanego działania z innymi.

Światła akomodacyjne radziły sobie szczególnie dobrze w uruchomieniach o znacznym natężeniu ruchu.

8. Podsumowanie

Ten rozdział zawiera podsumowanie całej pracy dyplomowej. Omówiono w nim realizację poszczególnych zadań i uzyskane rezultaty, które następnie skonfrontowano z założonym celem pracy. Opisano także potencjalne kierunki rozwoju oraz innych, wartych przeprowadzenia badań.

8.1. Wykonane prace

Przed przystąpieniem do właściwego projektowania rozwiązania, przeprowadzono dokładną analizę prac naukowych dotyczących uczenia ze wzmocnieniem, jego zastosowania w celu optymalizacji sterowania ruchem drogowym i innych wykorzystywanych algorytmów w tym celu. Zapoznano się z dziedziną zarządzania ruchem oraz towarzyszącymi jej, aktualnie stosowanymi w rzeczywistym świecie urządzeniami. Na podstawie dokonanego przeglądu wybrano niebadane wcześniej, perspektywiczne zagadnienia i zakreślono wstępnie projekt rozwiązania. Zaznajomiono się także z narzędziami, które są najczęściej wykorzystywane podczas badania podobnych zadań. Wyszukano również informacje na temat charakterystyk ruchu kołowego oraz układów skrzyżowań spotykanych na całym świecie.

Po wstępnym określeniu celu, aby głębiej zaznajomić się z charakterystyką problemu, stworzono prototyp realizujący podobne, uproszczone zadanie. Dzięki czasowi poświęconemu pracy nad nim oraz analizie jego działania, zauważono szczególne zagadnienia wymagające rozwiązania. Ich ominięcie byłoby bardzo problematyczne na późniejszym etapie, gdyby od razu przystąpiono do projektowania. Stworzono oraz wygenerowano kilkanaście siatek ulic wykorzystywanych następnie do uczenia i testowania najpierw prototypu, a potem także i projektowanego rozwiązania.

Najważniejszą napotkaną kwestią było pogodzenie ze sobą możliwości tworzonego rozwiązania do obsługi różnych skrzyżowań z ich znacznym zróżnicowaniu pod względem kształtu, jak i układu. W literaturze, w momencie badania tego zagadnienia spotykane były wyłącznie heterogeniczne pod względem architektury rozwiązania, korzystające z uczenia ze wzmocnieniem. Z tego powodu, przyłożono szczególną wagę do dokładnego zaprojektowania ostatecznej struktury projektu. Zdecydowano się na stworzenie agenta korzystającego z głębokiej sieci neuronowej, a także na jego powiązanie z pasem drogi dojazdowej do skrzyżowania, co nazwano agentem-pasem. Zgodnie z założeniami, podejście takie miało pozwolić na znaczną dowolność rodzajów i układów siatek ulic, na których można by je zastosować.

Następnie przystąpiono do implementacji rozwiązania. W tym celu, konieczne było zastosowanie kilku modyfikacji komponentów oryginalnego algorytmu wykorzystywanego do projektu. Zmodyfikowano bufor pamięci agenta, stworzono funkcję nagrody i przedstawiono sposób komunikacji pomiędzy poszczególnymi agentami-pasami oraz w ramach całego skrzyżowania. Za pomocą utworzonych wcześniej siatek ulic wykonano proces strojenia hiperparametrów. Następnie przystąpiono do zadania uczenia sieci neuronowej za pomocą algorytmu DDQN.

W wyniku przeprowadzonego procesu uczenia uzyskano sześć modeli sieci neuronowej. W celu dokładnego zweryfikowania działania stworzonego rozwiązania, poddano je różnorodnym testom. Uruchomiono serię kontrolnych scenariuszy, korzystając z siatek ulic używanych wcześniej w celu uczenia oraz specjalnie wygenerowanego do tego zadania nowego układu ulic. Za ich pomocą zebrano dokładne informacje dotyczące przejazdów przeprowadzonych przez każdy pojazd, zbadano także wymuszone czekaniem na przejazd kolejki samochodów. Uzyskane dane posłużyły następnie do porównania wykorzystanych kontrolerów.

8.2. Wnioski

Na podstawie przeprowadzonej analizy danych uzyskanych w procesie testowania, można stwierdzić, że osiągnięte wyniki były zgodne z założeniami pracy. Głównym celem niniejszej pracy nie było osiąganie jak najmniejszych czasów przejazdu samochodów, ale zaproponowanie rozwiązania, które byłoby w prosty sposób możliwe do zastosowania na skrzyżowaniach o różnych układach. Pomimo wyników nie pozwalających w definitywny sposób określić lepszego sposobu do sterowania światłami ruchu drogowego, podejście agenta-pasa jest perspektywiczne.

Uzyskane w ten sposób sterowanie ruchem było porównywalne względem akomodacyjnego kontrolera. Nauczone modele lepiej sprawdzały się na scenariuszach o niższych natężeniach ruchu. Kontroler akomodacyjny był wydajniejszy od agentów, gdy ruch uliczny był intensywny. Mimo to, oba porównywane kontrolery uzyskiwały podobne ilości najlepszych oraz najgorszych wyników, co świadczy o podobnej jakości zarządzania ruchem obu rozwiązań, z różnymi domyślnymi miejscami przeznaczenia zastosowania. W przypadku obu uruchomień na siatce „random” agenty zapewniały gorszą (uruchomienie 17) oraz o wiele gorszą (uruchomienie 18) jakość sterowania. Ponownie, mogło to wynikać z nadmiernego ruchu ulicznego. Innym powodem takiego zachowania mógł być brak przeszkolenia na podobnych skrzyżowaniach, co jednak nie powinno mieć miejsca, ponieważ agenty nie posiadały informacji na temat układu skrzyżowania (poza sąsiadami). Z pewnością, w przypadku dalszego rozwoju projektu przydatna byłaby dodatkowa weryfikacja tego zagadnienia. Agenty źle radziły sobie ze wzmożonym ruchem, ponieważ ciągle konkurowały z innymi o zasób jakim był dostęp do skrzyżowania. W celu usprawnienia ich działania można rozwinąć poziom komunikacji między nimi.

W niektórych przypadkach jakość sterowania zapewnionego przez nauczone modele znacznie się różniła. Mimo podobnego programu uczącego, także przebieg procesu uczenia charakteryzował się dużą zmiennością. Jednak do zaobserwowania możliwe były pewne schematy wahań wartości funkcji straty w miarę przebiegu procesu uczenia, sugerujące, że nie są one przypadkowe i wynikały ze stworzonego skryptu. Uczenie ze wzmocnieniem jest podatne na losowość, a wpływ ten jest dodatkowo spotęgowany przez zastosowanie wieloagentowego uczenia ze wzmocnieniem, a więc możliwe, że małe zmiany w programie uczenia przełożyły się na znaczne różnice późniejszego działania modeli.

Jak pokazała analiza średnich kolejek uzyskanych w scenariuszach testowych, nie zostały one optymalnie stworzone i nie pokrywały w równomiernym stopniu możliwych do zaist-

nienia sytuacji. W niektórych przypadkach, natężenie ruchu występujące w scenariuszach okazało się małe, duże i bardzo duże. Skutkowało to niezbadaniem jakości sterowania w warunkach umiarkowanej liczby samochodów. O ile na etapie ewaluacji rozwiązania nie stanowi to znacznego problemu, ponieważ dalej dostarcza cennych danych pozwalających na porównanie różnych podejść, tak źle zaprojektowane epizody uczące mogłyby znacznie zaszkodzić jakości ostatecznego rozwiązania. W przypadku powtarzania eksperymentów konieczne byłoby rozważenie, czy nie należy przeprojektować programu uczenia oraz testowania.

Początkowo zaplanowany czas trwania skryptów uczącego i testującego okazał się niewystarczający. O ile początkowe, prostsze siatki ulic były stale wykorzystywane podczas implementacji prototypu czy też strojenia hiperparametrów, tak bardziej zaawansowane scenariusze zastosowano dopiero podczas końcowej fazy uczenia. Ogromny wzrost czasu trwania pojedynczego epizodu oraz wymaganych zasobów sprzętowych spowodował, że część stworzonych siatek drogowych nie została wcale wykorzystana. Założono działanie agentów na różnorodnych skrzyżowaniach, także w ramach jednego środowiska, więc zupełny brak weryfikacji projektu w takich warunkach byłby nierzetelny. Znaczne wydłużenie czasu uruchomienia wymusiło jednak ograniczenie zakresu przeprowadzonych testów. Mimo zmniejszenia liczby rozpatrywanych siatek ulic, porównując z innymi pracami, przeprowadzona weryfikacja poruszała stosunkowo szeroki zakres scenariuszy. Testy nie mogły być jednak jeszcze bardziej ograniczone z powodu charakterystyki projektu. W przypadku powtórnej próby przeprowadzenia analizy projektu z pewnością wymagane byłoby zapewnienie większej liczby mocniejszych urządzeń. Okazało się, że posiadane urządzenia, na których włączane było stworzone rozwiązanie, posiadały niewystarczające podzespoły - głównie zabrakło pamięci RAM.

8.3. Krytyczne spojrzenie

Uczenie ze wzmocnieniem jest z natury bardzo długotrwałym procesem. Pomijając fazę projektową i implementacyjną, najpierw uruchomienia w ramach strojenia hiperparametrów, a następnie procesu uczenia potrafiły zająć kilka do kilkunastu dni, co łącznie przekłada się na kilkumiesięczny okres maksymalnego obciążenia komputera. Biorąc pod uwagę tak znaczny czas potrzebny do przeprowadzenia procesu uczenia, każda optymalizacja efektywności obliczeniowej jest niezwykle cenna, nawet pomimo stale rosnących możliwości dzisiejszych komputerów. Niestety, zaprezentowane w niniejszej pracy podejście agenta-pasa ogranicza możliwość zastosowania wielowątkowości. Ponieważ agenty bardzo mocno oddziaływały na siebie w ramach pojedynczego skrzyżowania, konieczna była ich sekwencyjna obsługa. Dodatkową, znaczną wadą zastosowanego podejścia było jego zapotrzebowanie na pamięć RAM. Projekt pokazuje, że przeprowadzanie uczenia i testowania jest najbardziej skuteczne na maszynach do tego przystosowanych pod względem sprzętowym. Wybór słabszych maszyn mógłbym skutkować znacznym wydłużeniem całego procesu. W przypadku ewentualnego dalszego rozwoju każda dodatkowa operacja wpłynie na całkowity czas wykonywania.

Innym aspektem wymagającym dokładniejszego przeanalizowania była zasadność zawarcia w procesie uczenia długo trwających epizodów uruchamianych na skomplikowanych siatkach ulic przy znacznym natężeniu ruchu. Możliwe, że obserwacje uzyskane w warunkach maksymalnego obciążenia były mało wartościowe, ponieważ polityki uzyskane w procesie uczenia na tych danych sugerowały, iż osiągnięto pewnego rodzaju maksymalny poziom efektywności. Taka sytuacja może sugerować, że dodatkowe wydłużenie czasu uczenia nie doprowadziłoby do poprawienia wydajności systemu. W takim przypadku, aby uzyskać poprawę w sposobie działania systemu należałoby rozwinąć zasady działania systemu, przykładowo przez rozbudowanie komunikacji między agentami oraz dodanie wspólnych celów optymalizujących ruch.

Biorąc pod uwagę wartości uzyskane z funkcji straty, istnieje możliwość, że program uczenia mógł być lepiej zaplanowany. Lepszym rozwiązaniem byłoby także zastosowanie adaptacyjnego zmieniania aktualnego epizodu, wraz ze stabilizowaniem się funkcji straty. Przez długi czas na początku procesu uczącego, agenty nie zostały poddawane wystarczająco trudnym zadaniom. Z drugiej strony, końcowy etap tego procesu nadmiernie obfitował w trudne sytuacje, co mogło doprowadzić do przesycenia agentów i zapomnienia wcześniej nauczonych informacji. Możliwe także, że czas, po którym epizod uczący był przymusowo kończony był za długi. Wskutek tego, wraz z każdym przedłużeniem zaplanowanego czasu epizodu, modele różniły się coraz bardziej między sobą. Dodatkowo, ta sytuacja powodowała znaczne wydłużenie rzeczywistego czasu uczenia. Było to szczególnie uciążliwe dla większych siatek ulic, gdy całość przetwarzania wymagała dużo większej liczby operacji.

8.4. Kierunki rozwoju

W celu lepszego przetestowania agenta-pasa, możliwe jest jego dokładne porównanie z innymi występującymi w literaturze rozwiązaniami. Aby osiągnąć jak najlepszą wydajność, warte wykorzystania byłyby podobne pola stanu. Były one dokładniejsze i przekazywały znacznie lepsze dane na temat środowiska. Dodatkowo, w celu dalszego poprawienia uzyskanego sterowania możliwe byłoby także zastosowanie bardziej zaawansowanego algorytmu uczenia ze wzmocnieniem [60], [80]. Inną opcją jest również zestawienie działania stworzonego rozwiązania do innych występujących w literaturze.

Biorąc pod uwagę długi czas trwania procesu uczenia i testowania stworzonego rozwiązania, warto rozpatrzeć usprawnienia, mające na celu zwiększenie ich wydajności. Jednym z pomysłów byłoby skorzystanie z wielowątkowości. Ponieważ w ramach każdego skrzyżowania konieczne było sekwencyjne wybieranie akcji, nie było możliwe zrównoleglenie procesu decyzyjnego. Jednakże, decyzje podejmowane na różnych skrzyżowaniach nie wpływały bezpośrednio na siebie więc istotną szansą byłoby stworzenie puli wątków i oddelegowanie operacji wyboru akcji w ramach każdej indywidualnej grupy agentów. Pozwoliłoby to także na ograniczenie liczby zawołań do sieci neuronowej, poprzez zebranie danych w ramach jednego wsadu od każdego aktualnie odpytywanego agenta.

Kolejną możliwością jest zastosowanie innego podejścia dotyczącego bufora *experience replay*. Wynikające ze znacznej liczby agentów duże zapotrzebowanie na pamięć RAM stanowiło największe ograniczenie wydajności. W celu minimalizacji tego wymagania

korzystna wydaje się centralizacja buforów poprzez stworzenie jednego centralnego o znacznym rozmiarze, zmniejszenie wielkości indywidualnych buforów agentów i przenoszenie zapisanych obserwacji co określony czas.

Istotną wadą agenta-pasa był brak jego możliwości do indywidualnego wystawienia różnych sygnałów w przypadku, gdy na jego pasie znajdował się więcej niż jeden kierunek, na przykład prawoskręt i jazda prosto. Takie podejście ograniczało elastyczność tworzonych programów świateł, ponieważ im większa liczba kierunków miała prawo przejazdu w danym momencie, tym większa liczba pozostałych kierunków była blokowana. Aby jak najbardziej zoptymalizować możliwość równoczesnego włączenia zielonego światła na wielu kierunkach, interesującą możliwością wydaje się powiązanie agenta z kierunkiem. Takie podejście z pewnością wymagać będzie bardziej rozwiniętego systemu komunikacji między agentami, jak również bardziej szczegółowych danych pochodzących na przykład od pojazdów IoT komunikujących się ze skrzyżowaniem. Obecnie, dostęp do tak dokładnych informacji nie jest jeszcze możliwy, jednak obszar ten jest przedmiotem intensywnych badań [81], [82].

Innym, ciekawym i z pewnością przydatnym kierunkiem badań byłaby analiza wpływu charakterystyki zadania i zastosowanego rozwiązania na optymalną liczbę kroków pomiędzy kolejnymi aktualizacjami wag sieci docelowej. Pomimo obszernych poszukiwań informacji na ten temat, nie odnaleziono żadnych wskazówek dotyczących optymalnego rzędu tej wielkości. Również autorzy prac naukowych rzadko kiedy zawierali tą daną w swoich artykułach. Dodatkowo, istniały jedynie mało istotne wytyczne poruszające strojenie tego hiperparametru, dlatego cenne byłyby publikacje dotyczące tej kwestii, szczególnie wraz z porównaniem ze sposobem aktualizacji wag *soft update* [59] oraz zbadaniem wpływu parametru na uczenie w środowisku wieloagentowym.

Bibliografia

[1] D. Braess, „Über ein Paradoxon aus der Verkehrsplanung”, *Unternehmensforschung*, t. 12, s. 258–268, 1968. adr.: https://api.semanticscholar.org/CorpusID:39202189.

[2] J. Douglass B. Lee, L. A. Klein i G. Camus, „Induced Traffic and Induced Demand”, *Transportation Research Record*, t. 1659, nr. 1, s. 68–75, 1999. DOI: 10.3141/1659-09. adr.: https://doi.org/10.3141/1659-09.

[3] T. Litman, „Generated Traffic and Induced Travel: Implications for Transport Planning”, *Institute of Transportation Engineers Journal*, t. 71, s. 38–47, kw. 2004.

[4] E. Commission i D.-G. for Environment, *Reclaiming city streets for people – Chaos or quality of life ?* Publications Office, 2004.

[5] TomTom, 2022. adr.: https://www.tomtom.com/traffic-index/warsaw-traffic/ (term. wiz. 09.10.2023).

[6] National Motor Museum. adr.: https://nationalmotormuseum.org.uk/ufaqs/when-were-the-first-traffic-lights-installed/ (term. wiz. 07.10.2023).

[7] C. Mcshane, „The Origins and Globalization of Traffic Control Signals”, *Journal of Urban History*, t. 25, s. 379–404, 1999. adr.: https://api.semanticscholar.org/CorpusID:110125733.

[8] G. F. Newell, „Memoirs on Highway Traffic Flow Theory in the 1950s”, *Operations Research*, t. 50, nr. 1, s. 173–178, 2002, ISSN: 0030364X, 15265463. adr.: http://www.jstor.org/stable/3088468 (term. wiz. 07.10.2023).

[9] Suchorzewski Konsulting. (2010), adr.: https://siskom.waw.pl/komunikacja/ZSZR/SW_ZSZR_Warszawa_II_i_III.pdf (term. wiz. 07.10.2023).

[10] D. Robertson i R. Bretherton, „Optimizing networks of traffic signals in real time-the SCOOT method”, *IEEE Transactions on Vehicular Technology*, t. 40, nr. 1, s. 11–15, 1991. DOI: 10.1109/25.69966.

[11] A. Sims i K. Dobinson, „The Sydney coordinated adaptive traffic (SCAT) system philosophy and benefits”, *IEEE Transactions on Vehicular Technology*, t. 29, nr. 2, s. 130–137, 1980. DOI: 10.1109/T-VT.1980.23833.

[12] A. Stevanovic, C. Kergaye i P. Martin, „SCOOT and SCATS: A Closer Look into Their Operations”, sty. 2009.

[13] P. Varaiya, „The Max-Pressure Controller for Arbitrary Networks of Signalized Intersections”, w *Advances in Dynamic Network Modeling in Complex Transportation Systems*, S. V. Ukkusuri i K. Ozbay, red. New York, NY: Springer New York, 2013, s. 27–66, ISBN: 978-1-4614-6243-9. DOI: 10.1007/978-1-4614-6243-9_2. adr.: https://doi.org/10.1007/978-1-4614-6243-9_2.

[14] J. Little, *The Synchronization of Traffic Signals by Mixed-Integer Linear Programming*. Creative Media Partners, LLC, 2015, ISBN: 9781340319762. adr.: https://books.google.pl/books?id=c0wjzwEACAAJ.

[15] Y. Dujardin, F. Boillot, D. Vanderpooten i P. Vinant, „Multiobjective and multimodal adaptive traffic light control on single junctions”, w *2011 14th International IEEE*

Conference on Intelligent Transportation Systems (ITSC), 2011, s. 1361–1368. DOI: 10.1109/ITSC.2011.6082977.

[16] Q. He, K. L. Head i J. Ding, „Multi-modal traffic signal control with priority, signal actuation and coordination", *Transportation Research Part C: Emerging Technologies*, t. 46, s. 65–82, 2014, ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2014.05.001. adr.: https://www.sciencedirect.com/science/article/pii/S0968090X14001144.

[17] D. Zhao, Y. Dai i Z. Zhang, „Computational Intelligence in Urban Traffic Signal Control: A Survey", *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, t. 42, nr. 4, s. 485–494, 2012. DOI: 10.1109/TSMCC.2011.2161577.

[18] F. Rasheed, K.-L. A. Yau, R. M. Noor, C. Wu i Y.-C. Low, „Deep Reinforcement Learning for Traffic Signal Control: A Review", *IEEE Access*, t. 8, s. 208 016–208 044, 2020. DOI: 10.1109/ACCESS.2020.3034141.

[19] J. Liu, J. Li, L. Zhang i in., „Secure intelligent traffic light control using fog computing", *Future Generation Computer Systems*, t. 78, s. 817–824, 2018, ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2017.02.017. adr.: https://www.sciencedirect.com/science/article/pii/S0167739X17302157.

[20] H. Marcos, R. Gernowo, I. Rosyida i O. D. Nurhayati, „Intelligent Traffic Management System using Internet of Things: A Systematic Literature Review", w *2022 IEEE 8th International Conference on Computing, Engineering and Design (ICCED)*, 2022, s. 1–6. DOI: 10.1109/ICCED56140.2022.10010602.

[21] P. Mannion, J. Duggan i E. Howley, „An Experimental Review of Reinforcement Learning Algorithms for Adaptive Traffic Signal Control", w *Autonomic Road Transport Support Systems*, T. L. McCluskey, A. Kotsialos, J. P. Müller, F. Klügl, O. Rana i R. Schumann, red. Cham: Springer International Publishing, 2016, s. 47–66, ISBN: 978-3-319-25808-9. DOI: 10.1007/978-3-319-25808-9_4. adr.: https://doi.org/10.1007/978-3-319-25808-9_4.

[22] H. Wei, G. Zheng, V. Gayah i Z. Li, *A Survey on Traffic Signal Control Methods*, 2020. arXiv: 1904.08117 [cs.LG].

[23] K.-L. A. Yau, J. Qadir, H. L. Khoo, M. H. Ling i P. Komisarczuk, „A Survey on Reinforcement Learning Models and Algorithms for Traffic Signal Control", *ACM Comput. Surv.*, t. 50, nr. 3, 2017, ISSN: 0360-0300. DOI: 10.1145/3068287. adr.: https://doi.org/10.1145/3068287.

[24] Aimsun, *Aimsun Next 23 User's Manual*, Aimsun Next 23.0.0, Barcelona, Spain, 2023. adr.: https://docs.aimsun.com/next/23.0.0/ (term. wiz. 30.01.2024).

[25] T. M. Inc., *MATLAB version: 9.13.0 (R2022b)*, Natick, Massachusetts, United States, 2022. adr.: https://www.mathworks.com.

[26] G. D. Cameron i G. I. Duncan, „PARAMICS—Parallel microscopic simulation of road traffic", *The Journal of Supercomputing*, t. 10, s. 25–53, 1996.

[27] P. A. Lopez, M. Behrisch, L. Bieker-Walz i in., „Microscopic Traffic Simulation using SUMO", w *The 21st IEEE International Conference on Intelligent Transportation Systems*, IEEE, 2018. adr.: https://elib.dlr.de/124092/.

[28] PTV Planung Transport Verkehr AG, *VISSIM [Software]*, Oprogramowanie do symulacji ruchu drogowego, 2024. adr.: https://www.ptvgroup.com/en/solutions/products/ptv-vissim/ (term. wiz. 30. 01. 2024).

[29] Saidallah, Mustapha, El Fergougui, Abdeslam i Elalaoui, Abdelbaki Elbelrhiti, „A Comparative Study of Urban Road Traffic Simulators", *MATEC Web Conf.*, t. 81, s. 05 002, 2016. DOI: 10.1051/matecconf/20168105002. adr.: https://doi.org/10.1051/matecconf/20168105002.

[30] TomTom, 2022. adr.: https://developer.tomtom.com/traffic-stats/documentation/api/introduction (term. wiz. 30. 01. 2024).

[31] Telraam. adr.: https://telraam.net/en/what-is-telraam (term. wiz. 30. 01. 2024).

[32] A. Loder, L. Ambühl, M. Menendez i K. Axhausen, „Understanding traffic capacity of urban networks", *Scientific Reports*, t. 9, list. 2019. DOI: 10.1038/s41598-019-51539-5.

[33] ZDM Warszawa. (2022), adr.: https://zdm-warszawa.maps.arcgis.com/apps/dashboards/fb6da3d215334a489709f7dc02726311 (term. wiz. 05. 01. 2024).

[34] R. Bąk, J. Chodur, M. Kieć, S. Gaca i K. Ostrowski, *Wytyczne projektowania skrzyżowań drogowych*. Ministerstwo Infrastruktury, Departament Dróg Publicznych, 2020. adr.: https://www.gov.pl/web/infrastruktura/wr-d.

[35] M. Tracz, J. Chodur, S. Gaca, S. Gondek, M. Kieć i K. Ostrowski, *Metoda obliczania przepustowości skrzyżowań z sygnalizacją świetlną*. Wydawnictwo PiT, 2004.

[36] J. Chodur, „Rozkład ruchu w grupie pasów na wlotach skrzyżowań z sygnalizacją", *Transport miejski i regionalny*, s. 3–12, list. 2014.

[37] J. W. Buckholz, *Introduction to Traffic Signal Phasing*. adr.: https://www.cedengineering.com/courses/introduction-to-traffic-signal-phasing (term. wiz. 31. 01. 2024).

[38] M. Kyte i T. Urbanik, „Traffic signal systems operations and design : an activity-based learning approach", 2012. adr.: https://www.webpages.uidaho.edu/TrafficSignalSystems/traffic/instructor/ch3a.pdf.

[39] R. Bąk, *Sygnalizacja świetlna na skrzyżowaniach*, 2020. adr.: https://wil.pk.edu.pl/index.php?option=com_rsfiles&task=rsfiles.download&path=spird%2FSSS_4-3_RB.pdf (term. wiz. 02. 02. 2024).

[40] lus-1. adr.: https://flic.kr/p/8mjiwP (term. wiz. 02. 04. 2024).

[41] Lifor. adr.: https://lifor.com.pl/produkty/fotoradary-stajconarne/traffistar-sr520/ (term. wiz. 02. 04. 2024).

[42] smartmicro. (2024), adr.: https://www.smartmicro.com/traffic-sensor (term. wiz. 31. 01. 2024).

[43] A. Guillen-Perez i M.-D. Cano, „Intelligent IoT systems for traffic management: A practical application", *IET Intelligent Transport Systems*, t. 2020, s. 1–13, sty. 2021. DOI: 10.1049/itr2.12021.

[44] K. Gao, S. Huang, J. Xie, N. N. Xiong i R. Du, „A Review of Research on Intersection Control Based on Connected Vehicles and Data-Driven Intelligent Approaches", *Electronics*, t. 9, nr. 6, 2020, ISSN: 2079-9292. DOI: 10.3390/electronics9060885. adr.: https://www.mdpi.com/2079-9292/9/6/885.

[45] A. Ławrynowicz, *Wprowadzenie do uczenia maszynowego*, 2017. adr.: https://www.cs.put.poznan.pl/alawrynowicz/SI_ML_2017_lawrynowicz_final.pdf.

[46] V. Dewanto i M. Gallagher, „Examining Average and Discounted Reward Optimality Criteria in Reinforcement Learning", w *AI 2022: Advances in Artificial Intelligence*, H. Aziz, D. Corrêa i T. French, red., Cham: Springer International Publishing, 2022, s. 800–813, ISBN: 978-3-031-22695-3.

[47] C. J. C. H. Watkins i P. Dayan, „Q-learning", *Machine Learning*, t. 8, nr. 3, s. 279–292, maj 1992, ISSN: 1573-0565. DOI: 10.1007/BF00992698. adr.: https://doi.org/10.1007/BF00992698.

[48] T. T. Nguyen, N. D. Nguyen i S. Nahavandi, „Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications", *CoRR*, t. abs/1812.11794, 2018. arXiv: 1812.11794. adr.: http://arxiv.org/abs/1812.11794.

[49] V. Mnih, K. Kavukcuoglu, D. Silver i in., *Playing Atari with Deep Reinforcement Learning*, 2013. arXiv: 1312.5602 [cs.LG].

[50] V. Mnih, K. Kavukcuoglu, D. Silver i in., „Human-level control through deep reinforcement learning", *Nature*, t. 518, s. 529–33, lut. 2015. DOI: 10.1038/nature14236.

[51] M. Roderick, J. MacGlashan i S. Tellex, *Implementing the Deep Q-Network*, 2017. arXiv: 1711.07478 [cs.LG].

[52] J. Kirkpatrick, R. Pascanu, N. Rabinowitz i in., „Overcoming catastrophic forgetting in neural networks", *Proceedings of the National Academy of Sciences*, t. 114, nr. 13, s. 3521–3526, 2017. DOI: 10.1073/pnas.1611835114. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1611835114. adr.: https://www.pnas.org/doi/abs/10.1073/pnas.1611835114.

[53] T. de Bruin, J. Kober, K. Tuyls i R. Babuska, „The importance of experience replay database composition in deep reinforcement learning", sty. 2015.

[54] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor i P. Stone, *Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey*, 2020. arXiv: 2003.04960 [cs.LG].

[55] L. Weng, „Curriculum for Reinforcement Learning", *lilianweng.github.io*, sty. 2020. adr.: https://lilianweng.github.io/posts/2020-01-29-curriculum-rl/.

[56] W. Zaremba i I. Sutskever, *Learning to Execute*, 2015. arXiv: 1410.4615 [cs.NE].

[57] H. Hasselt, „Double Q-learning", w *Advances in Neural Information Processing Systems*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel i A. Culotta, red., t. 23, Curran Associates, Inc., 2010. adr.: https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf.

[58] H. van Hasselt, A. Guez i D. Silver, „Deep Reinforcement Learning with Double Q-learning”, *CoRR*, t. abs/1509.06461, 2015. arXiv: 1509.06461. adr.: http://arxiv.org/abs/1509.06461.

[59] T. P. Lillicrap, J. J. Hunt, A. Pritzel i in., *Continuous control with deep reinforcement learning*, 2019. arXiv: 1509.02971 [cs.LG].

[60] M. Hessel, J. Modayil, H. van Hasselt i in., *Rainbow: Combining Improvements in Deep Reinforcement Learning*, 2017. arXiv: 1710.02298 [cs.AI].

[61] A. Ławrynowicz, *German Aerospace Center*, 2023. adr.: https://sumo.dlr.de/docs/Tutorials/OSMWebWizard.html.

[62] Martín Abadi, Ashish Agarwal, Paul Barham i in., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015. adr.: https://www.tensorflow.org/.

[63] A. Paszke, S. Gross, F. Massa i in., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019. arXiv: 1912.01703 [cs.LG].

[64] P. Fogliaroni, D. Bucher, N. Jankovic i I. Giannopoulos, „Intersections of Our World”, w *10th International Conference on Geographic Information Science (GIScience 2018)*, S. Winter, A. Griffin i M. Sester, red., ser. Leibniz International Proceedings in Informatics (LIPIcs), t. 114, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018, 3:1–3:15, ISBN: 978-3-95977-083-5. DOI: 10.4230/LIPIcs.GISCIENCE.2018.3. adr.: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.GISCIENCE.2018.3.

[65] R. Ducrocq i N. Farhi, „Deep Reinforcement Q-Learning for Intelligent Traffic Signal Control with Partial Detection”, *CoRR*, t. abs/2109.14337, 2021. arXiv: 2109.14337. adr.: https://arxiv.org/abs/2109.14337.

[66] T. Chu, J. Wang, L. Codecà i Z. Li, *Multi-Agent Deep Reinforcement Learning for Large-scale Traffic Signal Control*, 2019. arXiv: 1903.04527 [cs.LG].

[67] T. Wu, P. Zhou, K. Liu i in., „Multi-Agent Deep Reinforcement Learning for Urban Traffic Light Control in Vehicular Networks”, *IEEE Transactions on Vehicular Technology*, t. 69, nr. 8, s. 8243–8256, 2020. DOI: 10.1109/TVT.2020.2997896.

[68] X. Wang, L. Ke, Z. Qiao i X. Chai, „Large-Scale Traffic Signal Control Using a Novel Multiagent Reinforcement Learning”, *IEEE Transactions on Cybernetics*, t. 51, nr. 1, s. 174–187, 2021. DOI: 10.1109/TCYB.2020.3015811.

[69] P. LA i S. Bhatnagar, „Reinforcement Learning With Function Approximation for Traffic Signal Control”, *IEEE Transactions on Intelligent Transportation Systems*, t. 12, nr. 2, s. 412–421, 2011. DOI: 10.1109/TITS.2010.2091408.

[70] S. Gronauer i K. Diepold, „Multi-agent deep reinforcement learning: a survey”, *Artif. Intell. Rev.*, t. 55, nr. 2, s. 895–943, lut. 2022, ISSN: 0269-2821. DOI: 10.1007/s10462-021-09996-w. adr.: https://doi.org/10.1007/s10462-021-09996-w.

[71] V. Behzadan i A. Munir, *Vulnerability of Deep Reinforcement Learning to Policy Induction Attacks*, 2017. arXiv: 1701.04143 [cs.LG].

[72] I. Arel, C. Liu, T. Urbanik i A. G. Kohls, „Reinforcement learning-based multi-agent system for network traffic signal control", *IET Intelligent Transport Systems*, t. 4, nr. 2, s. 128–135, 2010.

[73] Z. Li, H. Yu, G. Zhang, S. Dong i C.-Z. Xu, „Network-wide traffic signal control optimization using a multi-agent deep reinforcement learning", *Transportation Research Part C: Emerging Technologies*, t. 125, s. 103 059, 2021, ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2021.103059. adr.: https://www.sciencedirect.com/science/article/pii/S0968090X21000851.

[74] D. Zha, K. Lai, K. Zhou i X. Hu, „Experience Replay Optimization", *CoRR*, t. abs/1906.08387, 2019. arXiv: 1906.08387. adr.: http://arxiv.org/abs/1906.08387.

[75] Y. Liu, L. Liu i W.-P. Chen, „Intelligent traffic light control using distributed multi-agent Q learning", w *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, s. 1–8. DOI: 10.1109/ITSC.2017.8317730.

[76] A. F. Agarap, „Deep Learning using Rectified Linear Units (ReLU)", *CoRR*, t. abs/1803.08375, 2018. arXiv: 1803.08375. adr.: http://arxiv.org/abs/1803.08375.

[77] W. Genders i S. N. Razavi, „Using a Deep Reinforcement Learning Agent for Traffic Signal Control", *CoRR*, t. abs/1611.01142, 2016. arXiv: 1611.01142. adr.: http://arxiv.org/abs/1611.01142.

[78] J. Zeng, J. Hu i Y. Zhang, „Adaptive Traffic Signal Control with Deep Recurrent Q-learning", w *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, s. 1215–1220. DOI: 10.1109/IVS.2018.8500414.

[79] D. Kingma i J. Ba, „Adam: A Method for Stochastic Optimization", *International Conference on Learning Representations*, grud. 2014.

[80] Z. Wang, N. de Freitas i M. Lanctot, „Dueling Network Architectures for Deep Reinforcement Learning", *CoRR*, t. abs/1511.06581, 2015. arXiv: 1511.06581. adr.: http://arxiv.org/abs/1511.06581.

[81] M. A. Vieira, M. Vieira, P. Louro, P. Vieira i A. Fantoni, „Vehicular Visible Light Communication for Intersection Management", *Signals*, t. 4, nr. 2, s. 457–477, 2023, ISSN: 2624-6120. DOI: 10.3390/signals4020024. adr.: https://www.mdpi.com/2624-6120/4/2/24.

[82] E. Ndashimye, S. K. Ray, N. I. Sarkar i J. A. Gutiérrez, „Vehicle-to-infrastructure communication over multi-tier heterogeneous networks: A survey", *Computer Networks*, t. 112, s. 144–166, 2017, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2016.11.008. adr.: https://www.sciencedirect.com/science/article/pii/S1389128616303826.

Spis rysunków

Spis tabel

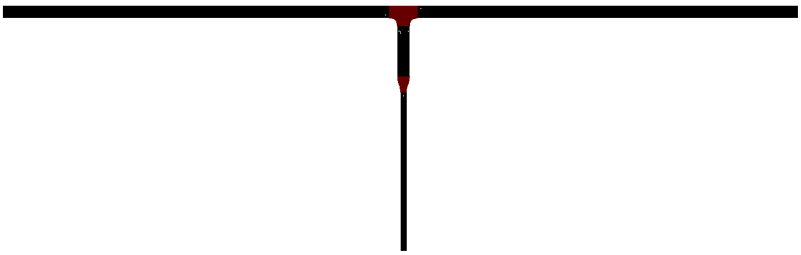
Spis załączników

Załącznik 1. Siatki ulic

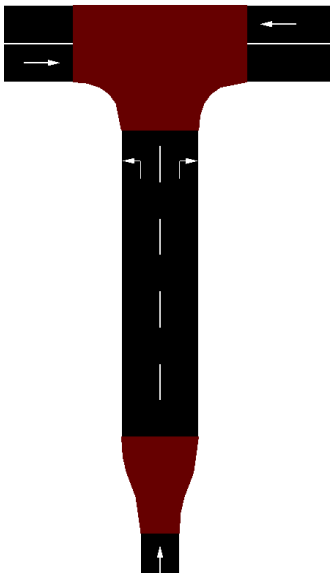
W tym załączniku przedstawiono wszystkie siatki ulic wykorzystywane w procesie uczenia oraz końcowego testowania stworzonego rozwiązania. Zostały podzielone na pięć grup, pod kątem podobieństwa oraz zbliżonej liczby agentów potrzebnych do uruchomienia scenariusza.

1.1. Grupa 1

1.1.1. T - skrzyżowanie z trzema drogami wlotowymi o kształcie „T”

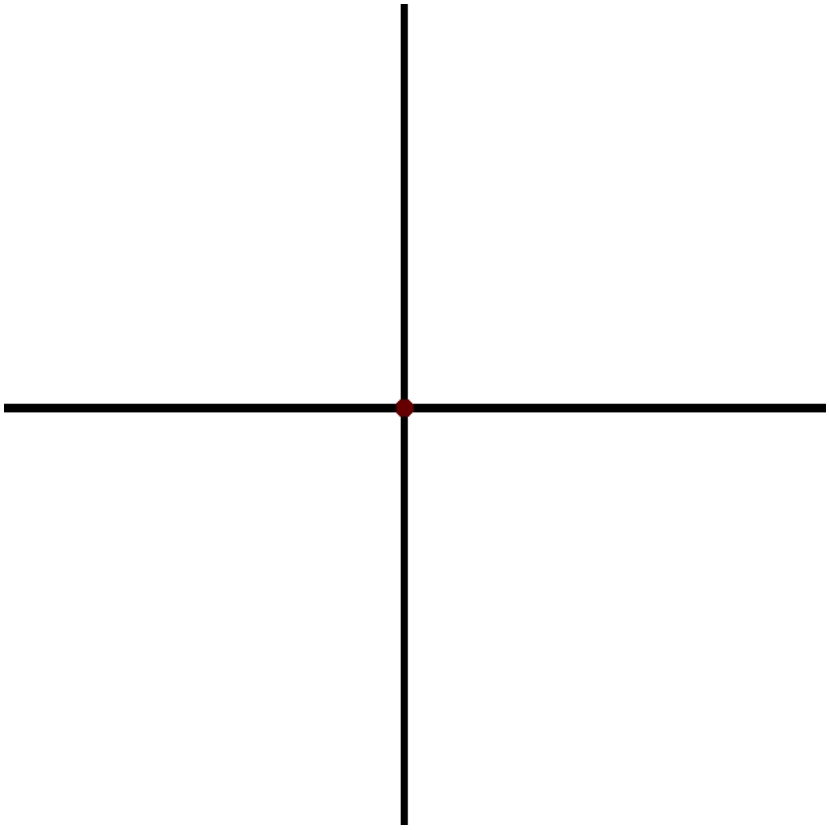


Rysunek 1.1. Widok na siatkę ulic

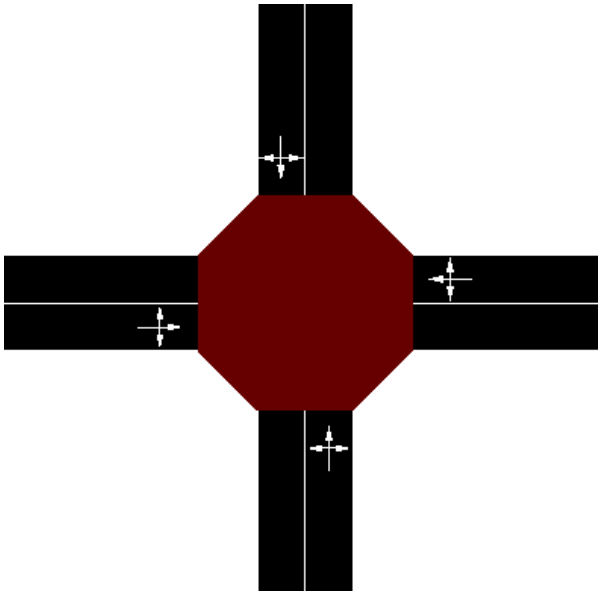


Rysunek 1.2. Widok z bliska na skrzyżowanie

1.1.2. 4x1 - 4 drogi wlotowe, każda z jednym pasem

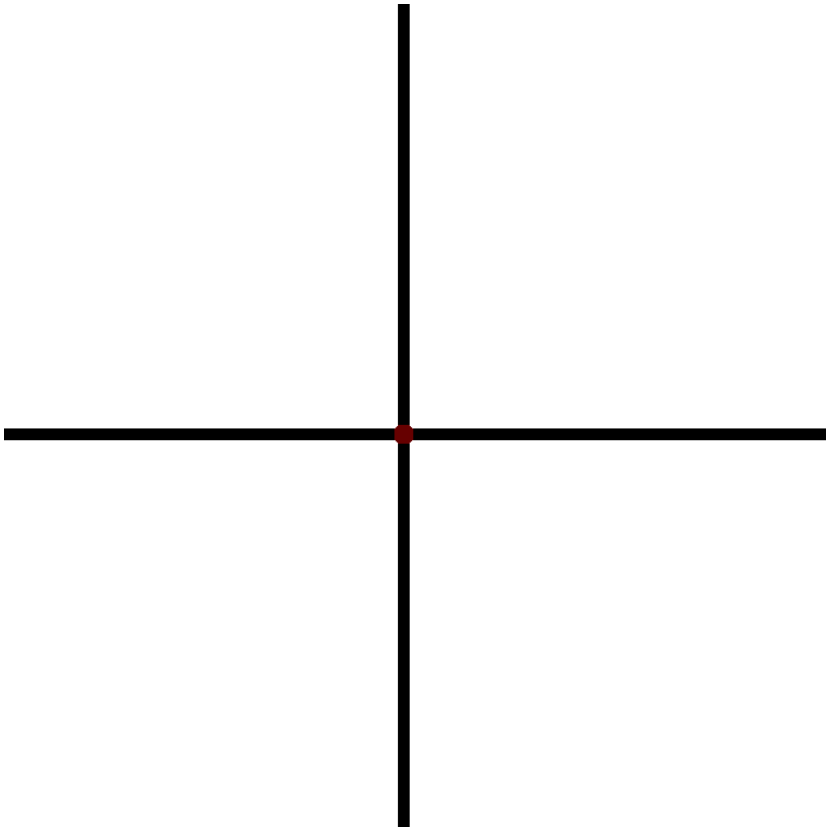


Rysunek 1.3. Widok na siatkę ulic

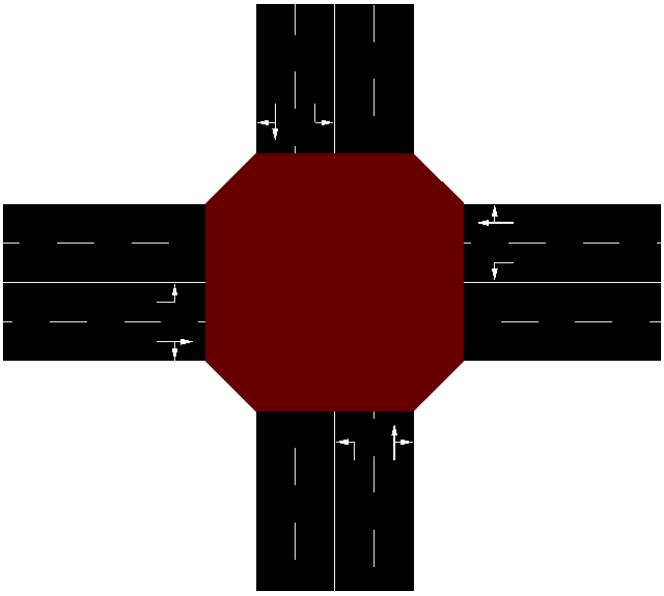


Rysunek 1.4. Widok z bliska na skrzyżowanie

1.1.3. 4x2



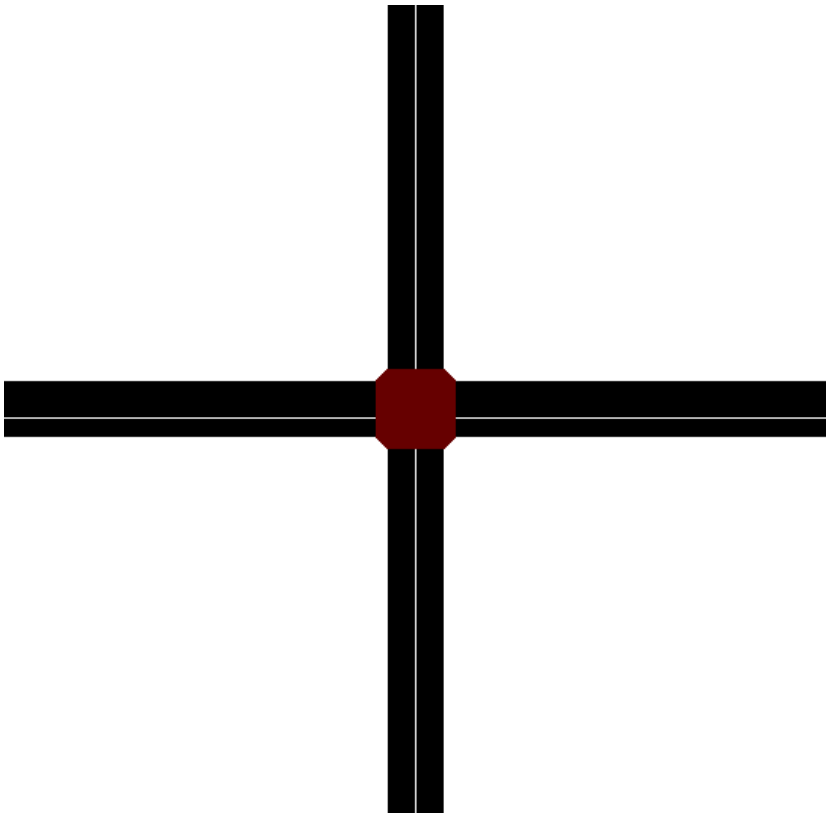
Rysunek 1.5. Widok na siatkę ulic



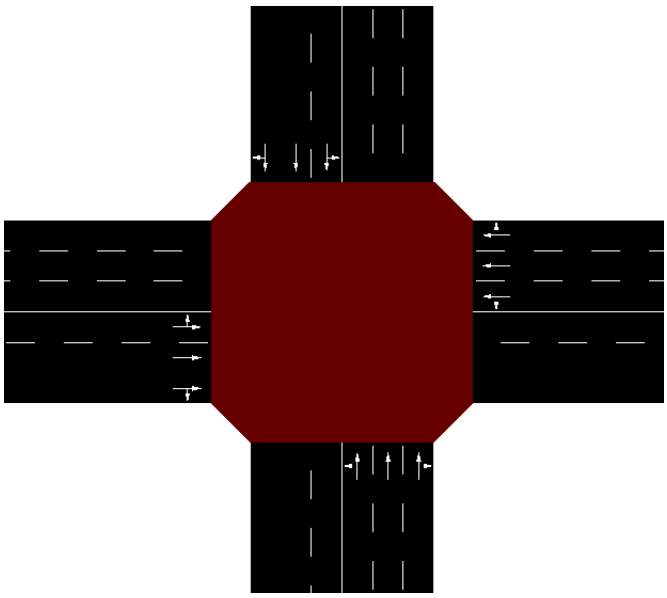
Rysunek 1.6. Widok z bliska na skrzyżowanie

1.2. Grupa 2

1.2.1. 4x3

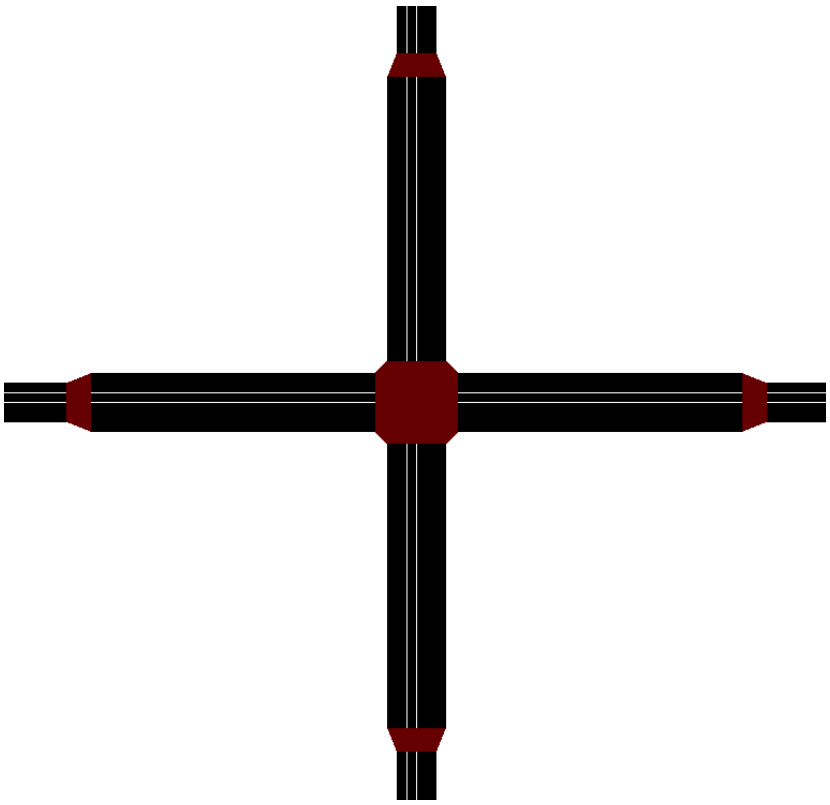


Rysunek 1.7. Widok na siatkę ulic

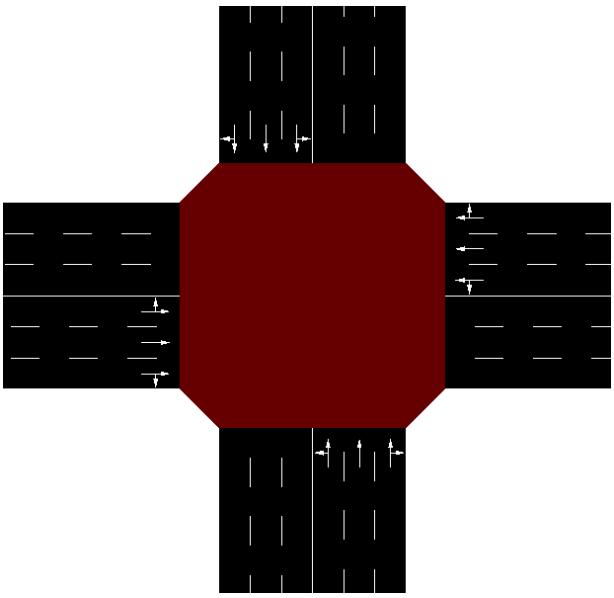


Rysunek 1.8. Widok z bliska na skrzyżowanie

1.2.2. 4x3 - węższe drogi dojazdowe

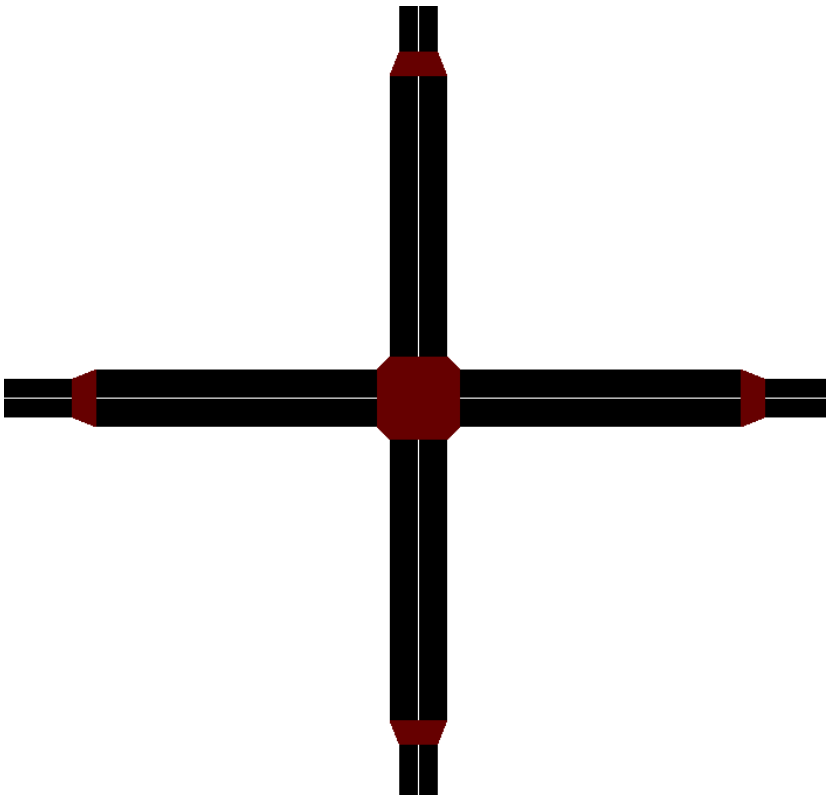


Rysunek 1.9. Widok na siatkę ulic

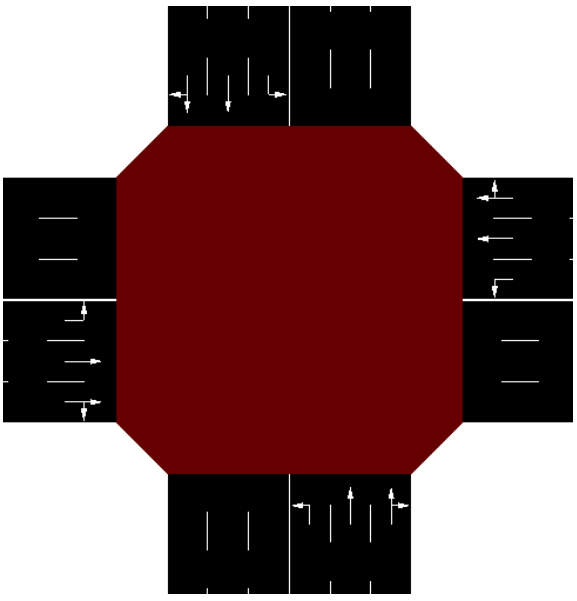


Rysunek 1.10. Widok z bliska na skrzyżowanie

1.2.3. 4x3 - węższe drogi dojazdowe, wyłączne lewoskręty

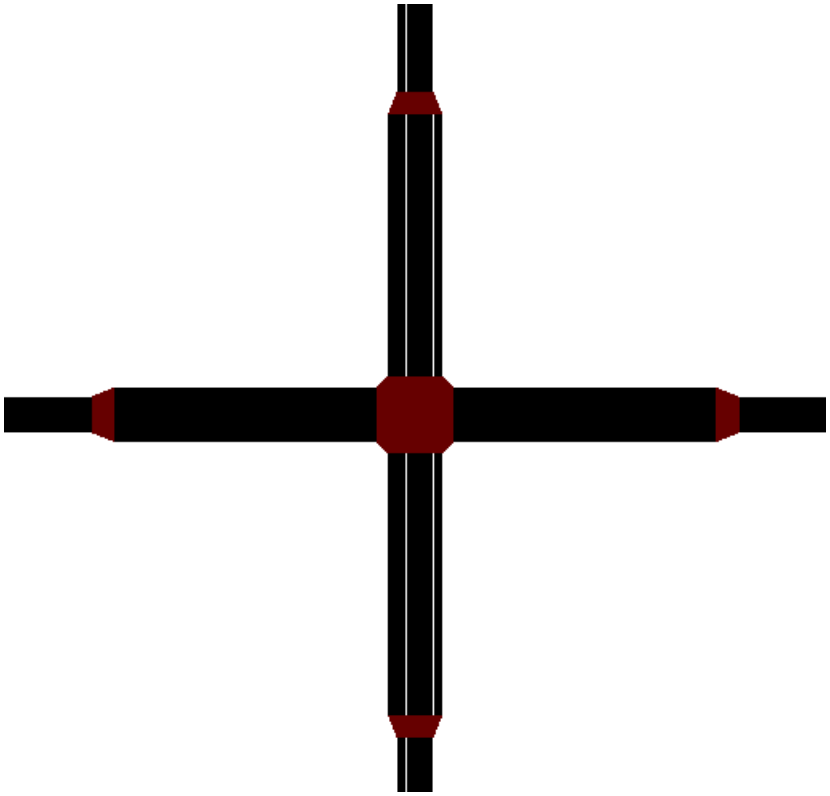


Rysunek 1.11. Widok na siatkę ulic

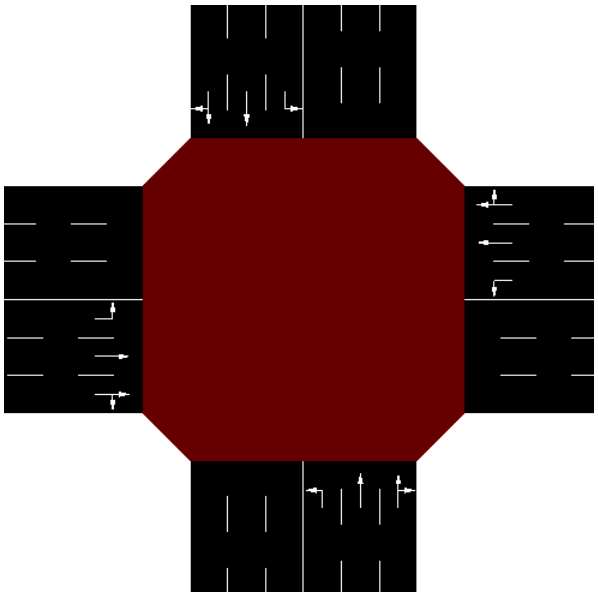


Rysunek 1.12. Widok z bliska na skrzyżowanie

1.2.4. 4x3 - węższe drogi dojazdowe, wyłączne lewoskręty, skrzyżowanie akomodacyjne z większą liczbą faz



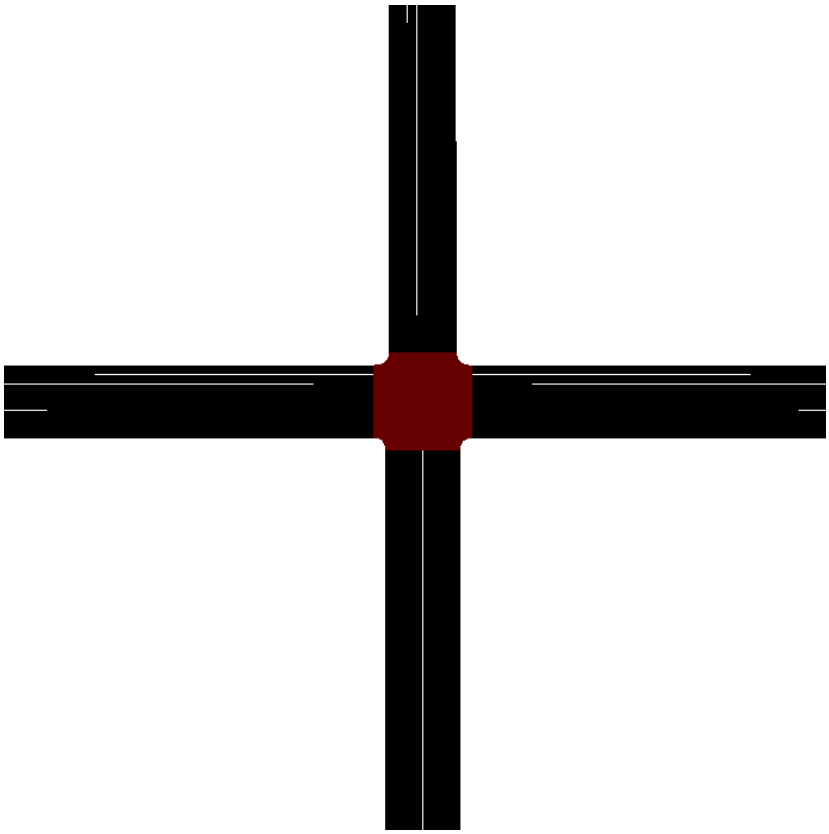
Rysunek 1.13. Widok na siatkę ulic



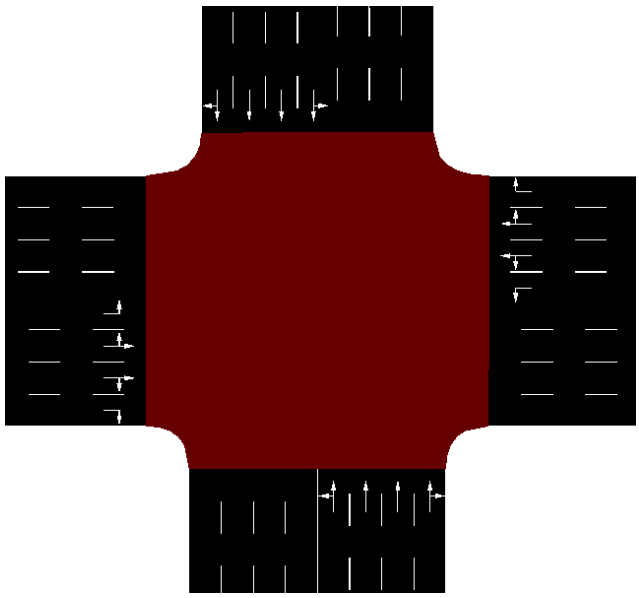
Rysunek 1.14. Widok z bliska na skrzyżowanie

1.3. Grupa 3

1.3.1. 4x4

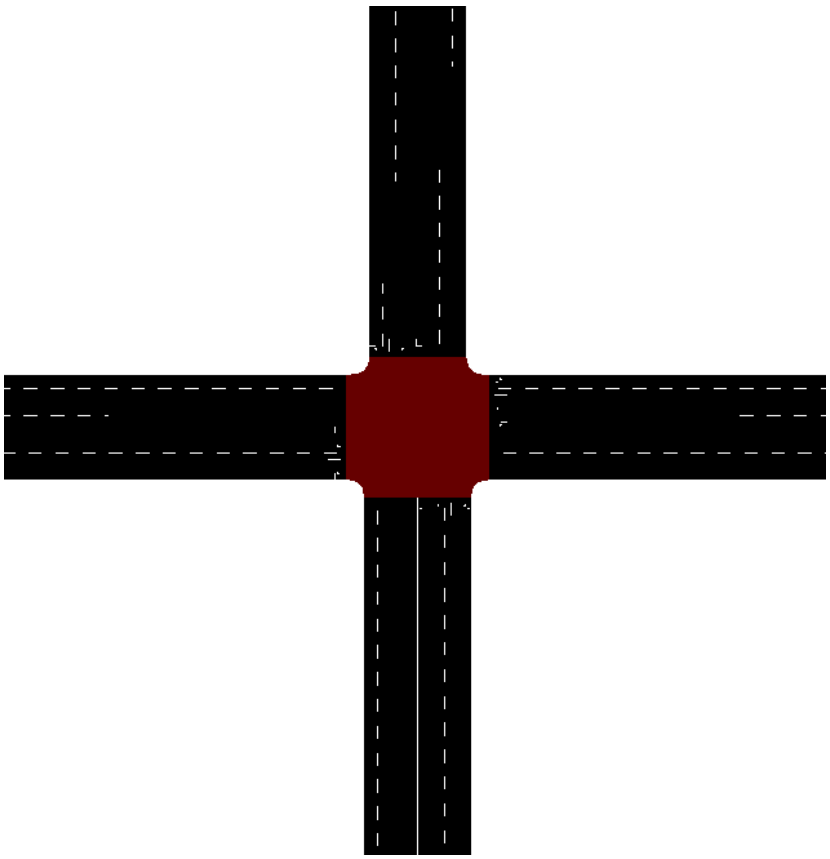


Rysunek 1.15. Widok na siatkę ulic

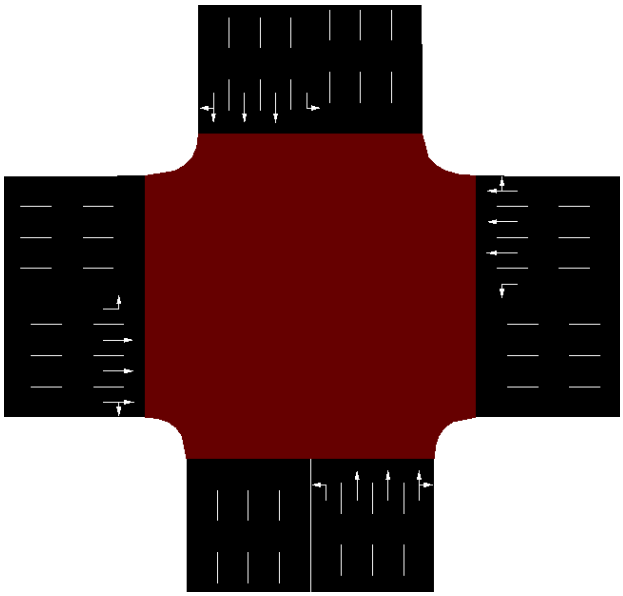


Rysunek 1.16. Widok z bliska na skrzyżowanie

1.3.2. 4x4 - skrzyżowanie akomodacyjne z większą liczbą faz

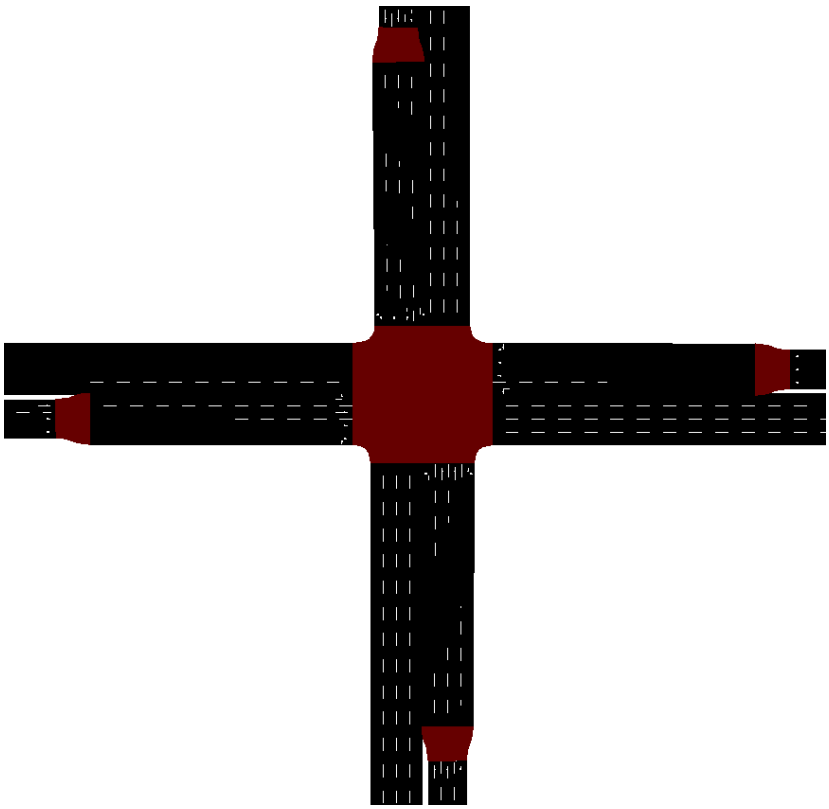


Rysunek 1.17. Widok na siatkę ulic

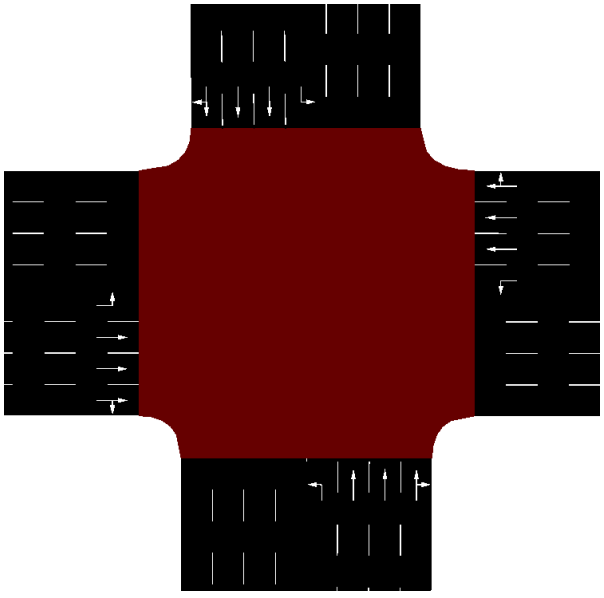


Rysunek 1.18. Widok z bliska na skrzyżowanie

1.3.3. 4x4 - węższe drogi dojazdowe, skrzyżowanie akomodacyjne z większą liczbą faz



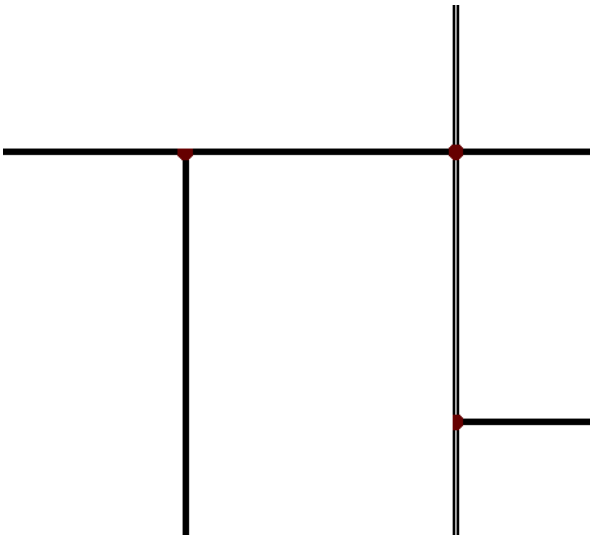
Rysunek 1.19. Widok na siatkę ulic



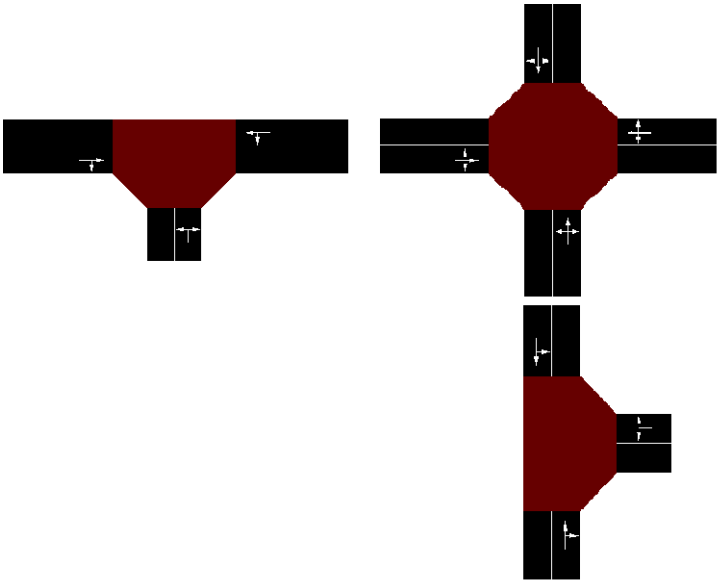
Rysunek 1.20. Widok z bliska na skrzyżowanie

1.4. Grupa 4

1.4.1. siatka1 - siatka jednopasmowych dróg składająca się z dwóch skrzyżowań trzywlotowych i jednego czterowlotowego

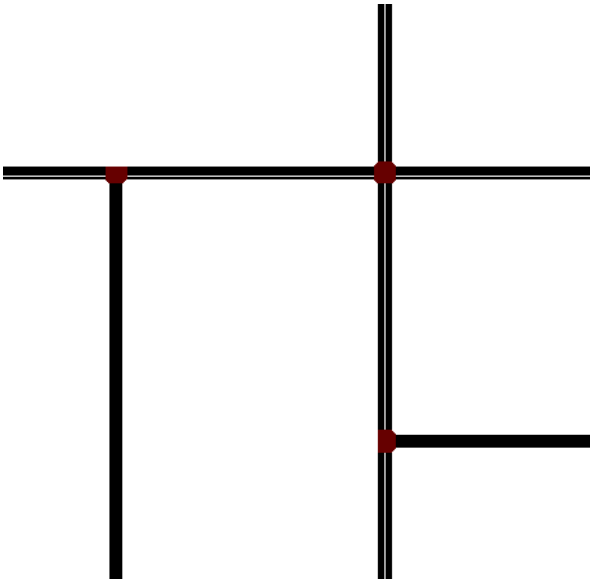


Rysunek 1.21. Widok na siatkę ulic

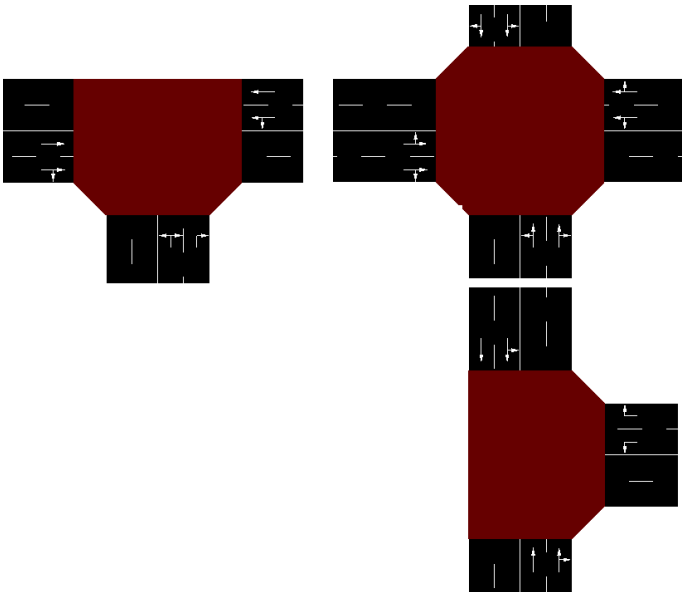


Rysunek 1.22. Widok z bliska na skrzyżowania występujące w siatce

1.4.2. siatka2 - siatka1, ale każda droga ma dwa pasy

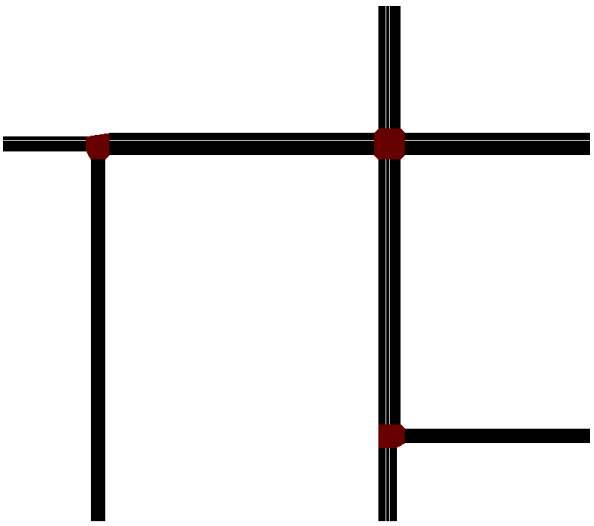


Rysunek 1.23. Widok na siatkę ulic

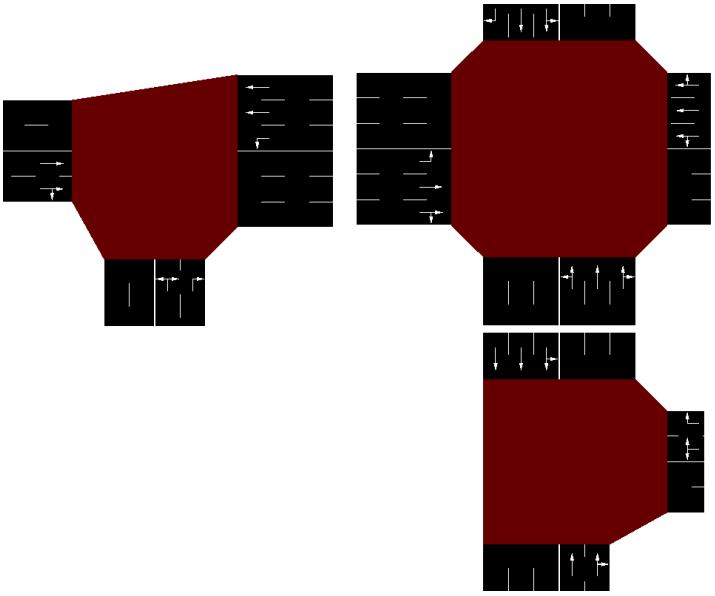


Rysunek 1.24. Widok z bliska na skrzyżowania występujące w siatce

1.4.3. siatka3 - siatka 2-3 pasmowych dróg



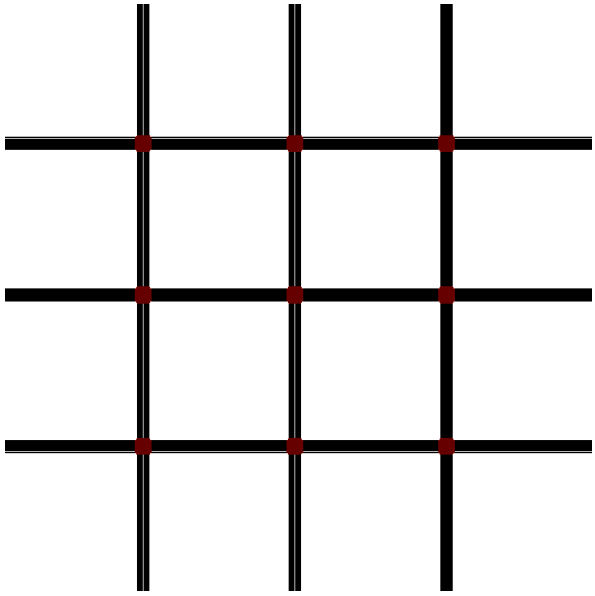
Rysunek 1.25. Widok na siatkę ulic



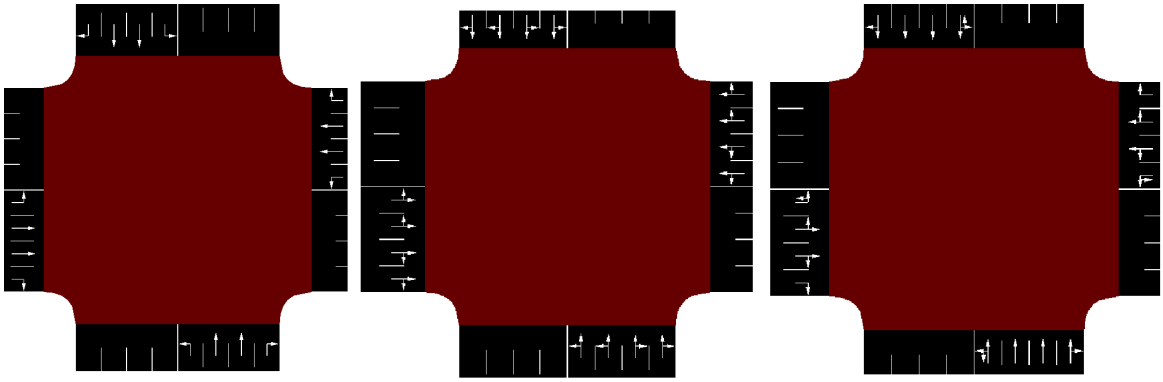
Rysunek 1.26. Widok z bliska na skrzyżowania występujące w siatce

1.5. Grupa 5

1.5.1. siatka3x3 - 9 równo rozłożonych skrzyżowań o różnych kierunkach dozwolonego przejazdu

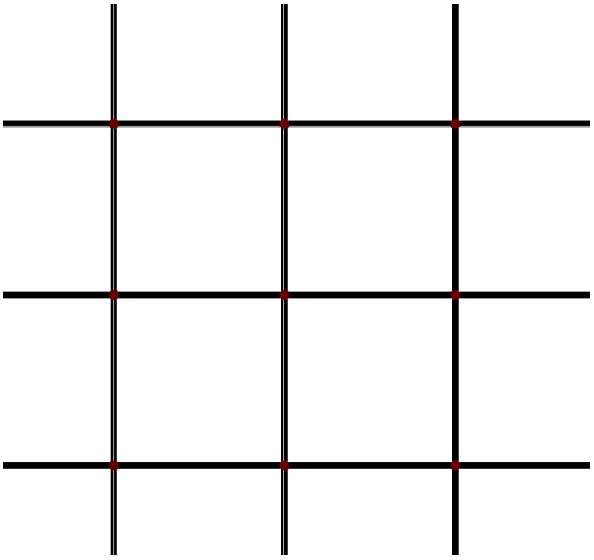


Rysunek 1.27. Widok na siatkę ulic

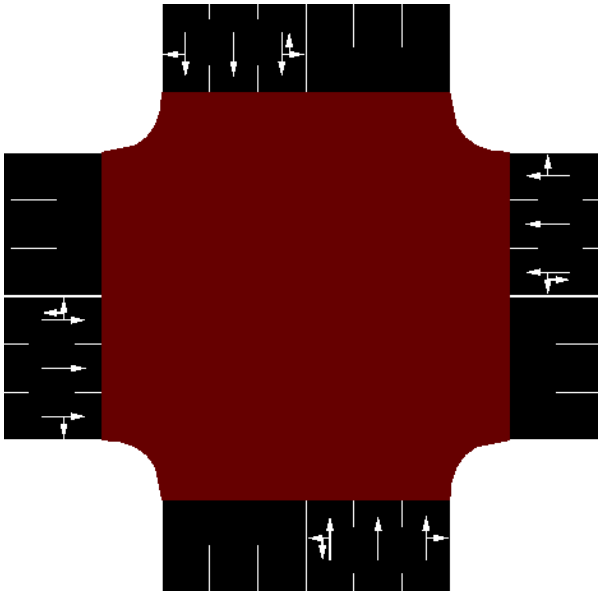


Rysunek 1.28. Widok z bliska na skrzyżowania występujące w siatce

1.5.2. siatka3x3-v2 - 9 równo rozłożonych skrzyżowań o identycznych kierunkach dozwolonego przejazdu

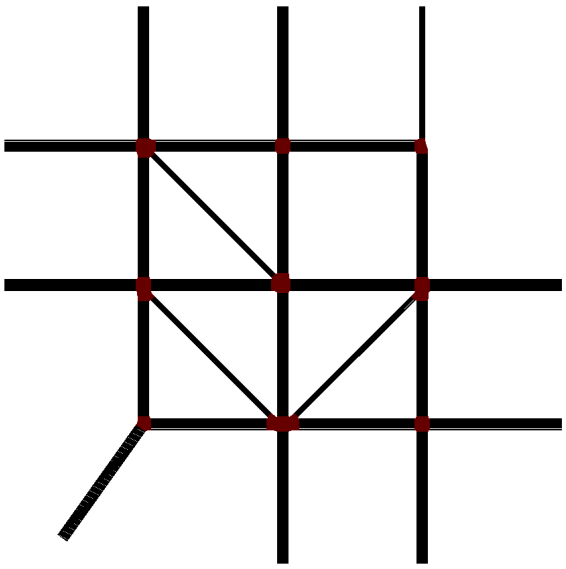


Rysunek 1.29. Widok na siatkę ulic

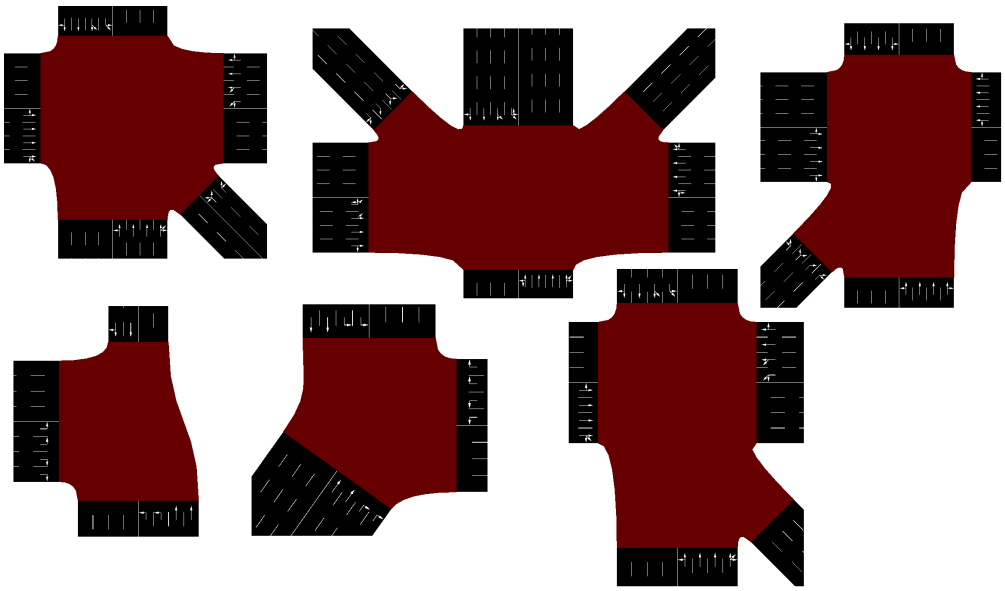


Rysunek 1.30. Widok z bliska na skrzyżowania występujące w siatce

1.5.3. vargrid - siatka ulic o różnej liczbie pasów (2-4) oraz skrzyżowań o różnej liczbie wlotów (3-5)



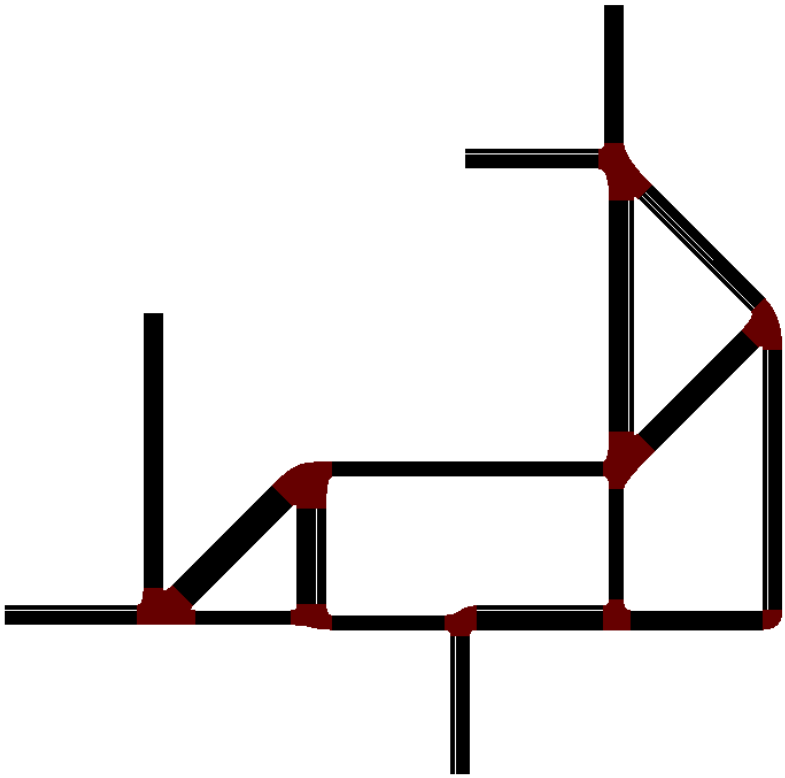
Rysunek 1.31. Widok na siatkę ulic



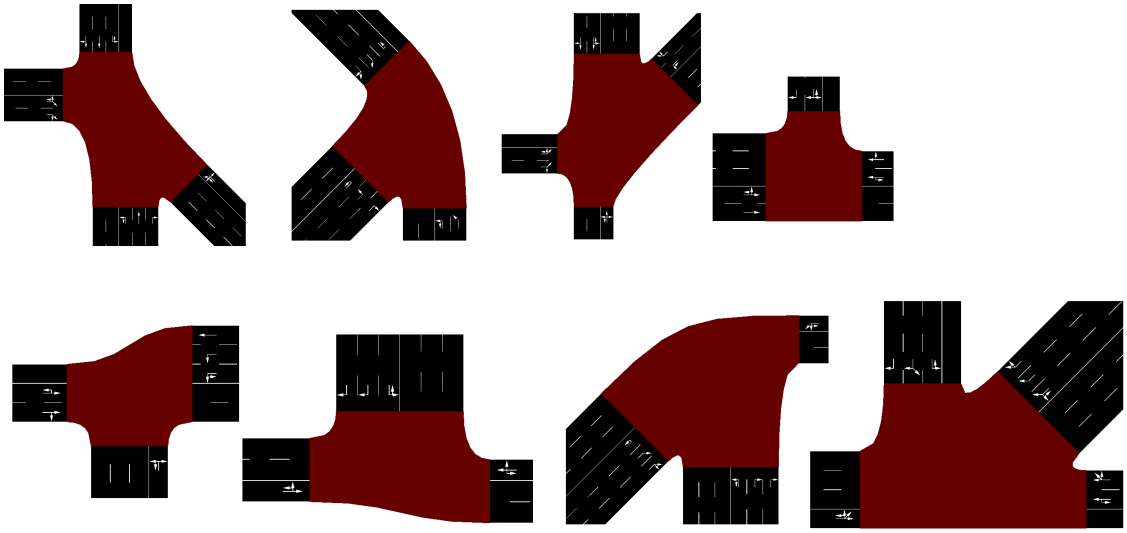
Rysunek 1.32. Widok z bliska na niektóre skrzyżowania występujące w siatce

1.6. Scenariusz testowy

1.6.1. random - losowa siatka ulic o 1-3 pasach i 8 skrzyżowaniach



Rysunek 1.33. Widok na siatkę ulic



Rysunek 1.34. Widok na siatkę ulic

Załącznik 2. Tabele wyników

W tym załączniku przedstawiono wszystkie obliczone dane statystyczne dla każdego z modeli uzyskanych w procesie uczenia.

Tabela 2.1. Wyniki testów dla siatki „T”

lp. (Liczba tras)	**Algorytm / model**	**Liczba ukończonych tras**	**Σ scz. [s]**	$\overline{\textbf{scz.}}$ **[s]**	$\sigma_{\overline{\textbf{scz.}}}$ **[s]**	**Med. scz. [s]**
1 (2442)	ak.	2442	1007	0,41	1,28	0,00
	m1	2442	3103	1,27	2,45	0,00
	m2	2442	2114	0,86	2,03	0,00
	m3	2442	2232	0,91	2,07	0,00
	m4	2442	1844	0,75	1,87	0,00
	m5	2442	1435	0,58	1,79	0,00
	m6	2442	1359	0,55	1,75	0,00
2 (3619)	ak.	3619	3368	0,93	2,55	0,00
	m1	3619	12059	3,33	14,98	0,00
	m2	3619	5120	1,41	2,72	0,00
	m3	3619	5433	1,50	2,96	0,00
	m4	3619	4954	1,37	2,83	0,00
	m5	3619	5140	1,42	3,64	0,00
	m6	3619	4778	1,32	3,69	0,00
3 (7225)	ak.	7225	91906	12,72	13,93	8,00
	m1	7225	86664	12,00	31,88	6,00
	m2	7225	33798	4,68	4,30	4,00
	m3	7225	43916	6,08	6,06	5,00
	m4	7225	35623	4,93	4,76	4,00
	m5	7225	101679	14,07	18,92	6,00
	m6	7225	101164	14,00	20,63	8,00

Tabela 2.2. Wyniki testów dla siatki „4x2”

lp. (Liczba tras)	**Algorytm / model**	**Liczba ukończonych tras**	**Σ scz. [s]**	**$\overline{\text{scz.}}$ [s]**	**$\sigma_{\overline{\text{scz.}}}$ [s]**	**Med. scz. [s]**
4 (2671)	ak.	2671	4481	1,68	5,26	0,00
	m1	2671	1987	0,74	2,25	0,00
	m2	2671	1892	0,71	2,36	0,00
	m3	2671	5295	1,98	4,44	0,00
	m4	2671	1424	0,53	2,2	0,00
	m5	2671	1494	0,56	2,15	0,00
	m6	2671	1141	0,43	1,83	0,00
5 (4027)	ak.	4027	18155	4,51	9,36	0,00
	m1	4027	8265	2,05	3,98	0,00
	m2	4027	7800	1,94	4,34	0,00
	m3	4027	14531	3,61	6,18	0,00
	m4	4027	6290	1,56	3,81	0,00
	m5	4027	7180	1,78	4,39	0,00
	m6	4027	6006	1,49	4,02	0,00
6 (8081)	ak.	8081	897939	111,12	103,16	93,00
	m1	8081	272471	33,72	20,05	31,00
	m2	8081	726902	89,95	85,98	59,00
	m3	8081	305281	37,78	27,66	34,00
	m4	8081	734800	90,93	84,90	65,00
	m5	8081	84577	10,47	12,79	6,00
	m6	8081	89609	11,09	13,33	7,00

Tabela 2.3. Wyniki testów dla siatki „4x3”

lp. (Liczba tras)	**Algorytm / model**	**Liczba ukończonych tras**	**Σ scz. [s]**	$\overline{\textbf{scz.}}$ **[s]**	$\sigma_{\overline{\textbf{scz.}}}$ **[s]**	**Med. scz. [s]**
7 (2857)	ak.	2857	12211	4,27	4,70	3,00
	m1	2857	1924	0,67	2,33	0,00
	m2	2857	2926	1,02	2,87	0,00
	m3	2857	8260	2,89	5,82	0,00
	m4	2857	1485	0,52	1,90	0,00
	m5	2857	1737	0,61	2,28	0,00
	m6	2857	1355	0,47	1,92	0,00
8 (4464)	ak.	4464	21172	4,74	4,94	3,00
	m1	4464	8442	1,89	4,18	0,00
	m2	4464	9279	2,08	4,51	0,00
	m3	4464	18250	4,09	7,01	0,00
	m4	4464	7476	1,67	4,06	0,00
	m5	4464	9339	2,09	5,07	0,00
	m6	4464	7591	1,70	4,28	0,00
9 (10056)	ak.	10056	178331	17,73	18,04	12,00
	m1	10056	164736	16,38	14,71	13,00
	m2	10056	128154	12,74	12,91	10,00
	m3	10056	167656	16,67	17,01	12,00
	m4	10056	120248	11,96	13,10	8,00
	m5	10056	115085	11,44	13,90	7,00
	m6	10056	123613	12,29	13,79	8,00

Tabela 2.4. Wyniki testów dla siatki „4x4”

lp. (Liczba tras)	**Algorytm / model**	**Liczba ukończonych tras**	**Σ scz. [s]**	$\overline{\textbf{scz.}}$ **[s]**	$\sigma_{\overline{\textbf{scz.}}}$ **[s]**	**Med. scz. [s]**
10 (2972)	ak.	2972	15594	5,25	6,44	2,50
	m1	2972	7636	2,57	4,91	0,00
	m2	2972	11149	3,75	7,65	0,00
	m3	2972	16349	5,50	10,76	0,00
	m4	2972	6067	2,04	5,16	0,00
	m5	2972	7764	2,61	7,30	0,00
	m6	2972	5894	1,98	5,75	0,00
11 (4735)	ak.	4735	30684	6,48	8,20	3,00
	m1	4735	24208	5,11	7,96	1,00
	m2	4735	28941	6,11	9,93	1,00
	m3	4735	37592	7,94	12,08	3,00
	m4	4735	22539	4,76	8,02	0,00
	m5	4735	28897	6,10	11,94	0,00
	m6	4735	24202	5,11	9,92	0,00
12 (11583)	ak.	11583	363561	31,39	68,06	16,00
	m1	11583	1205873	104,11	165,12	71,00
	m2	11583	1083250	93,52	221,58	28,00
	m3	11583	1401122	120,96	206,02	60,00
	m4	11583	1048939	90,56	222,99	24,00
	m5	11583	378011	32,63	33,38	23,00
	m6	11583	407958	35,22	33,10	27,00

Tabela 2.5. Wyniki testów dla siatki „siatka3”

lp. (Liczba tras)	**Algorytm / model**	**Liczba ukończonych tras**	**Σ scz. [s]**	$\overline{\textbf{scz.}}$ **[s]**	$\sigma_{\overline{\textbf{scz.}}}$ **[s]**	**Med. scz. [s]**
13 (3465)	ak.	3465	40298	11,63	12,38	8,00
	m1	3465	16504	4,76	5,80	3,00
	m2	3465	15515	4,48	6,98	1,00
	m3	3465	22081	6,37	9,73	2,00
	m4	3465	11957	3,45	6,34	0,00
	m5	3465	12891	3,72	6,94	0,00
	m6	3465	11034	3,18	6,50	0,00
14 (5615)	ak.	5615	85711	15,26	15,62	11,00
	m1	5615	48393	8,62	9,16	6,00
	m2	5615	45324	8,07	10,09	5,00
	m3	5615	61498	10,95	13,40	7,00
	m4	5615	42021	7,48	9,34	4,00
	m5	5615	47685	8,49	12,47	3,00
	m6	5615	40568	7,22	10,42	3,00
15 (14945)	ak.	12570	9608569	764,40	2126,60	79,00
	m1	5155	11288930	2189,90	4359,99	394,00
	m2	10728	6491613	605,11	972,10	305,00
	m3	1066	669897	628,42	2486,53	62,00
	m4	3624	2606656	719,28	1926,44	264,50
	m5	1019	868227	852,04	2970,36	67,00
	m6	567	522663	921,80	3002,89	39,00

Tabela 2.6. Wyniki testów dla siatek „vargrid” oraz „random”

lp. (Liczba tras)	**Algorytm / model**	**Liczba ukończonych tras**	**Σ scz. [s]**	$\overline{\textbf{scz.}}$ **[s]**	$\sigma_{\overline{\textbf{scz.}}}$ **[s]**	**Med. scz. [s]**
16 (4320)	ak.	4320	110274	25,53	18,46	23,00
	m1	4320	26298	6,09	7,68	4,00
	m2	4320	31606	7,32	10,37	3,00
	m3	289	1209426	4184,87	5228,44	0,00
	m4	4320	13354	3,09	6,01	0,00
	m5	4320	14854	3,44	6,97	0,00
	m6	4320	11866	2,75	5,98	0,00
17 (4320)	ak.	7715	85762	11,12	13,26	7,00
	m1	7715	169229	21,94	19,76	17,00
	m2	7715	207538	26,90	26,13	21,00
	m3	6668	390666	58,59	383,27	28,00
	m4	7715	167867	21,76	21,02	17,00
	m5	7715	174938	22,68	26,81	15,00
	m6	7715	164467	21,32	29,63	14,00
18 (27000)	ak.	23987	4968972	207,15	268,77	116,00
	m1	19019	2404338	126,42	125,86	90,00
	m2	6180	1943502	314,48	1058,42	91,00
	m3	1584	746053	470,99	2062,11	67,00
	m4	8353	1362875	163,16	643,75	75,00
	m5	208	541797	2604,79	4916,33	33,00
	m6	360	764280	2123,00	4382,10	46,50