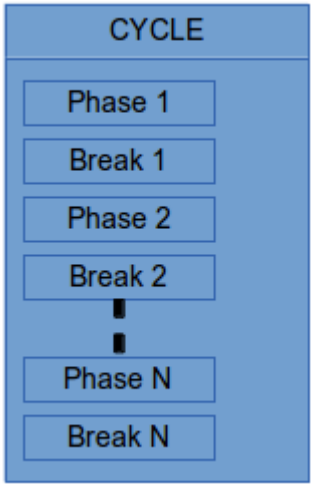
I. Wprowadzenie

Nikt już w dzisiejszych czasach nie zastanawia się nad potrzebą stosowania sygnalizacji świetlnej. Odkąd została ona zaadaptowana do sterowania ruchem drogowym, drastycznie zwiększyło się bezpieczeństwo przejazdu przez skrzyżowanie. Niestety, stało się to kosztem zmniejszenia przepustowości, a co za tym idzie wydłużenia czasu przejazdu z punktu A do punktu B, większej emisji spalin, podwyższeniu poziomu stresu u kierowców.

W celu zapewnienia bezpieczeństwa na skrzyżowaniu stosuje się cykle podzielone na fazy. Ich liczba zależy od charakterystyki skrzyżowania. W przypadku, gdy uczestnikami ruchu są rowerzyści, tramwaje, autobusy i piesi, liczba faz gwałtownie wzrasta w celu zapewnienia wszystkim możliwości bezpiecznego przejścia. Wadą takiego rozwiązania jest zatrzymanie ruchu na określony czas, aby pojazdy pozostające jeszcze na skrzyżowaniu mogły je opuścić oraz aby piesi przebywający jeszcze na pasach mogli z nich bezpiecznie zejść. Takie przerwy w ruchu są niezbędne do zapewnienia bezpieczeństwa. Im więcej faz w danym cyklu, tym więcej takich przerw.

Załóżmy, że cykl składa się z N faz. Między każdą z nich musi nastąpić przerwa w ruchu, która pozwala na opuszczenie skrzyżowania przez wszystkich uczestników ruchu, którzy wjechali na nie jeszcze w poprzedniej fazie. Dodatkowo załóżmy, że czas trwania takiej przerwy wynosi T sekund. Stąd otrzymujemy równanie określające całkowity czas, w którym przejazd jest zabroniony dla wszystkich uczestników ruchu:

$$Całkowity\,czas = N \times T \quad (1)$$



Ilustracja 1: Cykl składa się z faz oraz przerw, które są wykorzystywane w celu opuszczenia skrzyżowania przez wszystkich uczestników ruchu.

Przy założeniu, że liczna cykli się nie zmienia (N = const), czas bezczynności skrzyżowania wzrasta liniowo względem T we wzorze (1). **Istnieje zatem potrzeba minimalizowania funkcji (1) bez negatywnego wpływu na bezpieczeństwo przejazdu przez skrzyżowanie.**

Nie jest to jednak jedyny przypadek, gdy skrzyżowanie jest w stanie bezczynności. Bardzo łatwo

jest wyobrazić sobie sytuację, gdy zielone światło jest zapalone dla pasa ruchu, na którym nie ma żadnego pojazdu. W takim przypadku pożądanym zachowaniem byłoby uruchomienie kolejnej fazy bez przetrzymywania skrzyżowania w stanie nieproduktywnym.

Rozważmy proste skrzyżowanie przy pewnym ruchu ulicznym. Załóżmy, że wciągu całego cyklu światło zielone świeci przez czas T_b (indeks „b" został użyty dla podkreślenia czasu przebywania skrzyżowania w stanie bezczynności) dla pustych pasów dojazdowych. W skrajnych przypadkach ten czas może okazać się bardzo duży, sięgający nawet całkowitemu czasowi trwania cyklu. Z drugiej strony, w przypadku rozłożonego ruchu na wszystkie pasy dojazdowe, czas ten może być równy zero. **Dobry system sterujący ruchem drogowym na skrzyżowaniu, powinien minimalizować czas bezczynności, to jest dbać o to, by wartość T_b była jak najmniejsza.**

Kolejnym ważnym aspektem, który wpływa negatywnie na przepustowość skrzyżowania jest nieodpowiednie zachowanie kierowców oraz pieszych. Oczywiście algorytm sterowania ruchem drogowym ma ograniczone możliwość, jeżeli chodzi o eliminację tych zachowań. Może natomiast próbować minimalizować ich konsekwencje poprzez różne mechanizmy.

Do niepożądanych zachowań kierowców, które bardzo negatywnie wpływać na przepustowość skrzyżowania należą głównie:

1. Wjazd na skrzyżowanie bez upewnienia się, że można je bezpiecznie opuścić (taka sytuacja nie pozwala kolejnym pojazdom na przejechanie przez skrzyżowanie nawet w przypadku, gdy mają zielone światło i ich pas wylotowy jest wolny)
2. Zbyt wolna reakcja kierowców ruszających na zielonym świetle. Zbyt długie oczekiwanie, roztargnienie czy też brak wystarczających umiejętności zmniejszają liczbę pojazdów, które przejadą przez skrzyżowanie
3. Zmiana pasa ruchu w ostatniej chwili. Takie zachowanie powoduje wstrzymanie ruchu na pasie, który zajmuje pojazd chcący zmienić pas, oraz zwolnienie ruchu na pasie, który ten pojazd chce zająć

Inteligentny system powinien zatem nie pozwolić na wjazd na skrzyżowanie w sytuacji, gdy pas wyjazdowy jest przepełniony. Powinien także minimalizować skutki działania tak zwanej „fali" (powolnego rozpoczynania ruchu przez uprzednio stojące na czerwonym świetle pojazdy) poprzez eliminację zbyt krótkich faz w cyklu.

Istnieją także czynniki, na które algorytm nie będzie miał wpływu, takie jak wypadki, remonty, czy też wspomniana, nagła zmiana pasa przez kierującego. **Takie losowe elementy mają jednak miejsce i system powinien umieć je wykrywać i wprowadzać odpowiednią strategię, dopasowaną do zaistniałej sytuacji.**

Ludzkie zdrowie jest bezcenne. Może przekonać się o tym ten, któremu kiedyś go zabrakło. **Nasz system wspiera wszelkiego rodzaju pojazdy uprzywilejowane poprzez uruchamiane tryby alarmowe. Dzięki nim pojazdy uprzywilejowane będą mogły bezkolizyjnie przejechać przez skrzyżowanie w bardzo krótkim czasie.**

II. Przegląd rozwiązań

Zaimplementowanie dobrze działającego systemu jest zadaniem bardzo trudnym, które wymaga znajomości już istniejących rozwiązań. Ten rozdział ma na celu przedstawienie kilku podejść do realizacji systemu sterującego ruchem na skrzyżowaniu. Dzięki przeprowadzonej analizie, będziemy mieli świadomość wad i zalet proponowanych rozwiązań oraz będziemy mogli skorzystać z doświadczeń ludzi, którzy zajmowali się podobnym tematem.

Bardzo trudno jest porównywać poszczególne algorytmy. Zwykle autorzy porównują wyniki swoich rozwiązań do cyklów standardowych. Niestety, cykle te mają różne długości, a ruch testowy charakteryzuje się innymi parametrami. Uniemożliwia to prawie całkowicie porównanie różnych algorytmów. Jedyną informacją, którą jesteśmy w stanie wyciągnąć jest fakt, że algorytm okazał się o X procent lepszy od standardowego.

Pomimo tych trudności, chcemy jednak przedstawić te rozwiązania, które wydają nam się wartościowe zarówno ze względu na otrzymane rezultaty jak i samą koncepcję oraz sposób realizacji.

1. Systemy wieloagentowe

W naszej pracy skupimy się na sterowaniu ruchem na pojedynczym, niezależnym skrzyżowaniu. Chcielibyśmy jednak zaznaczyć, że wielu autorów zajmuje się interakcją wielu skrzyżowań, dzięki której problem sterowania jest rozwiązywany globalnie. Najczęstszym podejściem jest zainstalowanie na każdym ze skrzyżowań niezależnego algorytmu, który będzie pracował do chwili, gdy sytuacja stanie się na tyle skomplikowana, że problem będzie musiał zostać przeniesiony na wyższy poziom, to jest będzie musiał być rozpatrzony makroskopowo.

Aby jednak można było przejść do implementacji systemu wieloagentowego należy wcześniej opracować i przetestować algorytm, który będzie mógł sterować niezależnie każdym ze skrzyżowań w sieci. To właśnie jest celem tej pracy. Implementacja systemu, który będzie inteligentnie sterował ruchem drogowym na skrzyżowaniu oraz będzie w stanie stwierdzić, że sytuacja jest na tyle zła, że problem nie może być rozwiązany lokalnie. Nasza praca obejmuje zatem część systemu wieloagentowego, a dokładnie jednego agenta. Nie dotyczy jednak w żaden sposób komunikacji między agentami.

Ponieważ nasza praca obejmuje część większego systemu, chcielibyśmy w tym rozdziale przedstawić ogólną teorię dotyczącą systemów wieloagentowych.

Nie od dzisiaj wiadomo, że efektywna współpraca może przynieść bardzo wiele korzyści. Nawet w przyrodzie możemy znaleźć przykłady kooperacji, dzięki której grupy osobników są w stanie osiągnąć zamierzone cele w szybki i prosty sposób. Przykładem mogą być mrówki, które informują się nawzajem o lokalizacji pożywienia. [1]

Są trzy rodzaje komunikacji między agentami [1]:

1. Agenci mogą przesyłać sobie komunikaty o tak zwanych sensacjach (wydarzeniach, które mają znaczenie z perspektywy sterowania ruchem drogowym), akcjach oraz nagrodach natychmiastowo.
2. Agenci mogą komunikować się okresowo, przesyłając sekwencje składające się z już wspomnianych sensacji, akcji oraz nagród.
3. Agenci mogą wymieniać się wiedzą na temat strategii działania. Oznacza to, że agenci mogą wymieniać się wiedzą, którą nabyli w procesie uczenia.

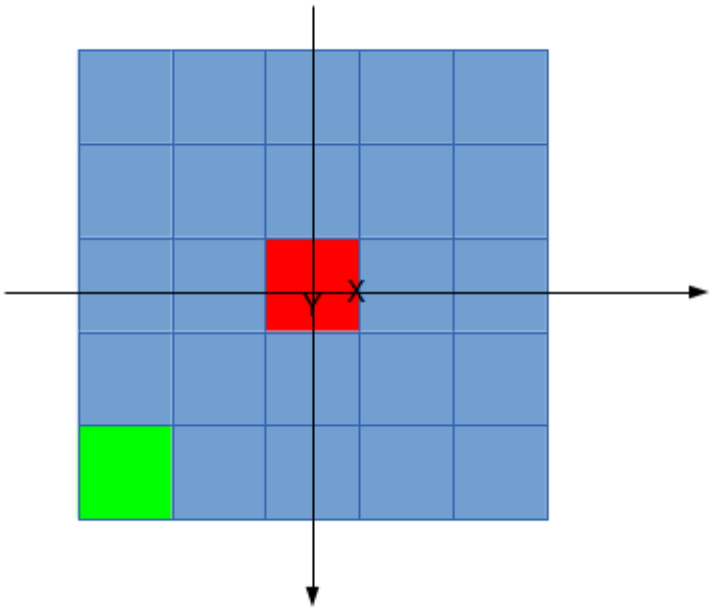
Dla zobrazowania rodzajów komunikacji między agentami posłużymy się przykładem agentów-łowców oraz agenta-ofiary [1].

Agenci-łowcy mają za zadanie złapać agenta-ofiarę (ofiara i łowca znajdują się w tej samej komórce) na polu o rozmiarze 10 na 10 jednostek. Ofiara porusza się w sposób losowy wewnątrz kwadratu. W każdej iteracji agent może wykonać jeden z czterech ruchów:

- ruch w górę
- ruch w dół

- ruch w lewo
- ruch w prawo

Oczywiście ruch nie może być wykonany, gdy jego efektem będzie przekroczenie granic kwadratu. Zakładamy dodatkowo, że wielu agentów może zajmować w tym samym czasie jedną komórkę. Agenci uczą się polować na ofiarę poprzez nagrody i kary. Za każdy ruch, po którym ofiara zostanie ujęta, agent otrzymuje nagrodę w wysokości +1. Ruch, który nie prowadzi do pola z ofiarą jest karany wartością -0.1. Agenci uczą się jak złapać ofiarę poprzez minimalizację sumy otrzymywanych wzmocnień (w tym wypadku kary lub nagrody). Każdy z łowców ma określone pole widzenia. Dla przykładu przyjmujemy, że jest ono równe kwadratowi o rozmiarze 2 na 2.



Ilustracja 2: Łowca jest oznaczony kolorem czerwonym. Ofiara (kolor zielony) jest w zasięgu jego możliwości percepcyjnych.

Pomijamy na razie wszelkie szczegóły, ponieważ działanie metody Q-Learning zostanie przedstawione szczegółowo w specjalnie do tego przeznaczonym rozdziale. W tym momencie chodzi nam tylko o ideę i rezultaty jakie można osiągnąć, gdy agenci ze sobą współpracują.

Proces uczenia polega na losowych ruchach agentów. Proces ten składa się z tak zwanych epizodów, które zaczynają się, gdy pierwszy agent wykonuje pierwszy ruch, a kończą się, gdy jeden z łowców stanie na polu z ofiarą. Liczba takich epizodów jest uzależniona od wybranych strategii uczenia. Poprzez takie działanie, agenci uczą się jak szybko i efektywnie znaleźć ofiarę.

Autor [1] przedstawił w tabeli swoje rezultaty po 200 epizodach. Jest to porównanie losowo poruszających się agentów z tymi, które uczyły się ze wzmocnieniem bez współpracy (każdy agent uczył się niezależnie).

Liczba ofiar / Liczba łowców	1/1	1/2
Losowo poruszający się agenci	123.08	56.47
Uczący się agenci	25.32	12.21

Tabela 1: Porównanie średniej liczby kroków wykonanych przez agentów od początku do złapania ofiary przez jednego z nich. Średnia jest oparta na 200 epizodach. Źródło [1].

Z tabeli wynika, że agenci, którzy przeszli proces szkolenia radzą sobie dużo lepiej z szukaniem ofiary. Rezultaty są o 492% lepsze w przypadku jednego łowcy i jednej ofiary oraz o 462% lepsze w przypadku jednej ofiary i dwóch łowców. Jest to pierwszy dowód na to, że nauczanie ze wzmocnieniem działa i daje bardzo dobre rezultaty.

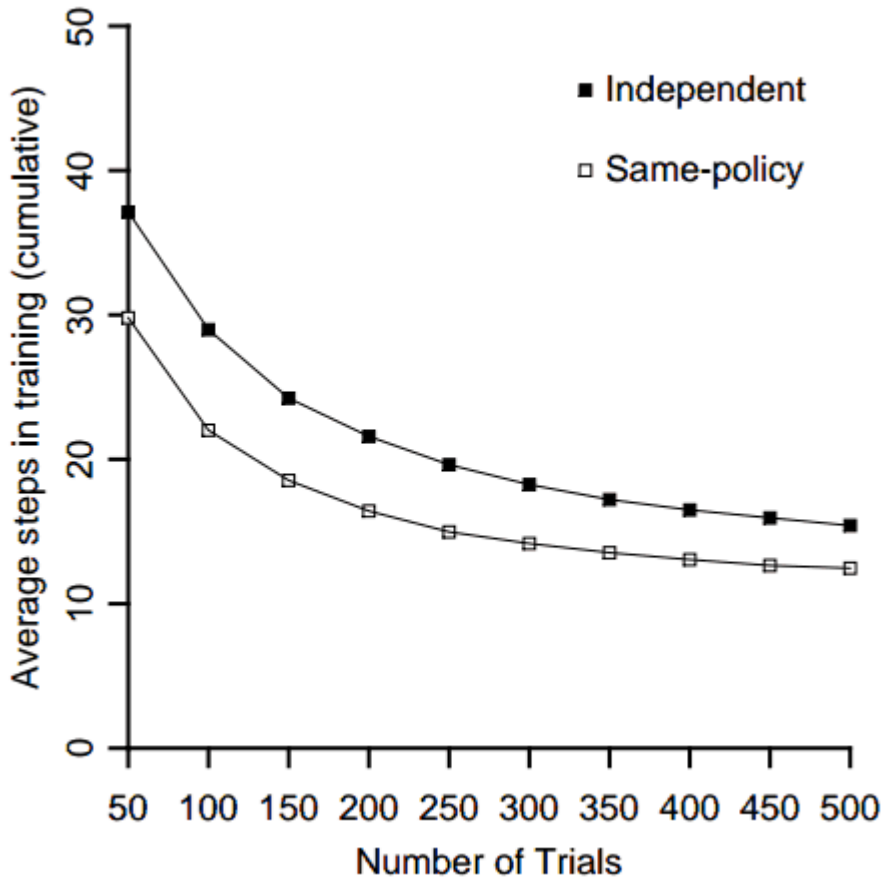
Kolejnym krokiem w badaniach autora zawartych w pracy [1], było opracowanie komunikacji między agentami, która mogłaby przyspieszyć proces uczenia. Agenci stali się teraz nie tylko łowcami, ale także informatorami. Gdy w polu widzenia agenta pierwszego pojawiła się ofiara, agent drugi był o tym informowany. Jest to tak zwane dzielenie się sensacją, czyli informacją istotną dla efektywnego wykonania zadania. Takie podejście doprowadziło autora do następujących rezultatów.

	Pole widzenia	Średnia liczba kroków do czasu złapania ofiary po 2000 epizodów	
		Trening	Test
Niezależni agenci	2x2	20.38	24.04
Współpracujący agenci	2x2	25.20	24.52
Niezależni agenci	3x3	14.65	16.04
Współpracujący agenci	3x3	14.02	12.98
Niezależni agenci	4x4	12.21	11.53
Współpracujący agenci	4x4	11.05	8.83

Tabela 2: Tabela przedstawia porównanie agentów, którzy pracowali niezależnie z agentami, którzy ze sobą współpracowali. Tabela przedstawia wyniki dla konfiguracji: jedna ofiara, dwóch łowców. Źródło [1.]

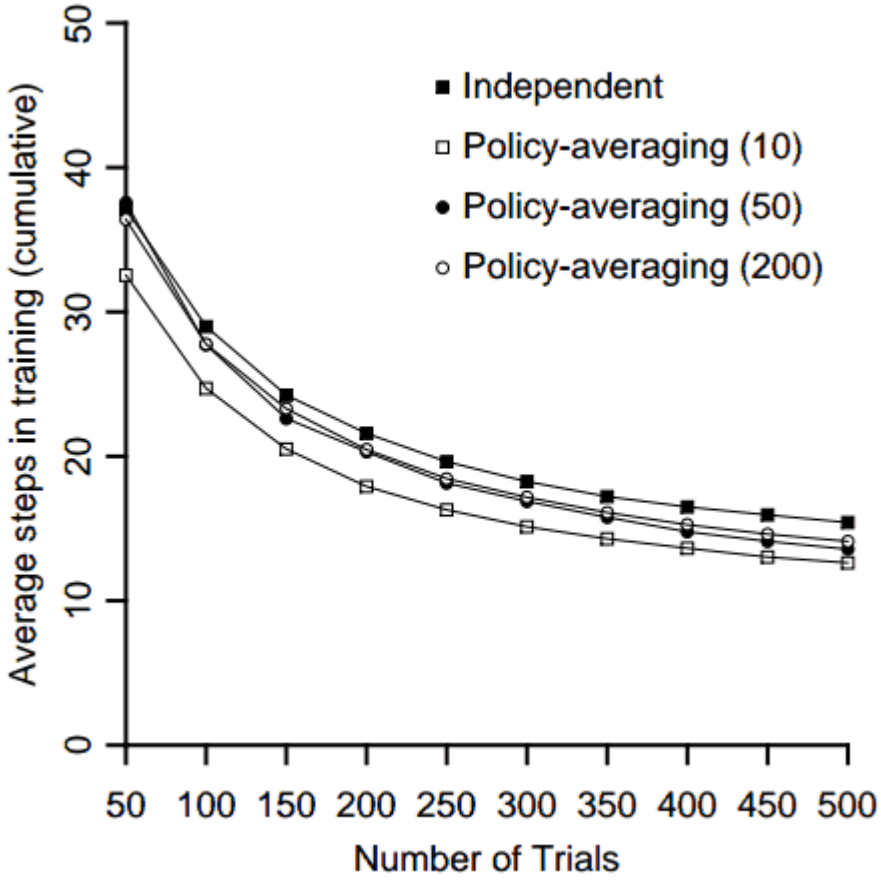
Z tabeli można wyciągnąć wniosek, że znaczną różnicę w efektywności uzyskuje się dopiero po zwiększeniu pola widzenia. W przypadku małych obszarów percepcji różnice w teście okazują się niewielkie. Możemy jednak zauważyć, że agenci, którzy wymieniają się między sobą sensacjami są zdolni do szybszego osiągnięcia określonego celu.

Autor rozważa także w swojej pracy [1] inny sposób transferu wiedzy między agentami. Opiera się on na założeniu, że agenci będą gromadzili całą swoją wiedzę w jednym miejscu. Oznacza to tyle, że zamiast implementacji niezależnych struktur dla każdego z agentów do przechowywania informacji o dotychczasowych postępach w nauce, wykorzystuje się tylko jedną, którą aktualizują wszyscy agenci. Takie podejście przyspiesza proces uczenia się agentów. Porównanie niezależnie działających agentów do agentów współdzielących bazę wiedzy przedstawione zostało na wykresie.



Rysunek 1: Wykres pokazuje porównanie agentów, którzy uczyli się niezależnie (independent) oraz agentów współdzielących się wiedzą (same policy). Źródło [1].

Dodatkowo autor [1] rozszerzył koncepcję wspólnej nauki o uśrednianie wiedzy agentów co pewien określony czas. Kolejny wykres pokazuje wyniki dla agentów niezależnych oraz agentów, których wiedza była uśredniana co 10, 50 oraz 200 kroków.



Rysunek 2: Porównanie czterech rodzajów agentów: niezależnych, z uśrednianiem co 10, 50 oraz 200 kroków. Źródło [1].

Korzystając z uśredniania jesteśmy w stanie niewiele przyspieszyć proces uczenia. Możemy jednak zaobserwować, że we wszystkich przypadkach w nieskończoności następuje zbieżność. Należy więc rozważyć, czy narzut spowodowany dodatkową operacją uśredniania jest warty tej niewielkiej różnicy w szybkości.

W naszej pracy będziemy chcieli zastosować propozycje uczenia, które przedstawił autor w publikacji [1]. Sprawdzimy, jak takie rozwiązania wpływają na szybkość i jakość uczenia się algorytmu.

W przypadku sterowania ruchem, bardzo trudno jest mówić o uśrednianiu bazy wiedzy między dwoma agentami, którzy są zainstalowani na skrzyżowaniach charakteryzujących się innymi parametrami. Stąd będziemy wykorzystywali to samo skrzyżowanie, z tak samo zdefiniowanym ruchem do uczenia algorytmu.

Pierwszy sposób na przyspieszenie uczenia i poprawę jego jakości będzie się opierał na wymianie sensacji. Sensację definiujemy, jako wybranie takiego zachowania, które doprowadzi do znacznego oczyszczenia skrzyżowania z pojazdów.

W kolejnych próbach będziemy utrzymywali podobnie jak autor [1] wspólną bazę wiedzy dla obu skrzyżowań, by następnie przetestować eksperymentalnie uśrednianie wiedzy i jej wpływ na proces nauczania. Wszystkie szczegóły oraz wyniki będą przedstawione w stosownych rozdziałach.

Jak już wcześniej wspomnieliśmy, będziemy się zajmować w niniejszej pracy optymalizacją sterowania ruchem na pojedynczym skrzyżowaniu. Nasza praca ma stanowić fundament przyszłych badań nad systemami wieloagentowymi, w których zostanie użyty nasz algorytm. Nie można bowiem tworzyć wielkiego systemu wielagentowego bez wcześniejszego opracowania

pojedynczego agenta oraz upewnienia się, że spełnia on powierzone mu zadanie. Taki właśnie jest cel naszej pracy.

W rzeczywistym mieście agenci (skrzyżowania) będą mogły się ze sobą wymieniać informacjami. Taka wymiana może opierać się o różne założenia, ale najczęściej w literaturze znajduje się następujące architektury systemów wieloagentowych.

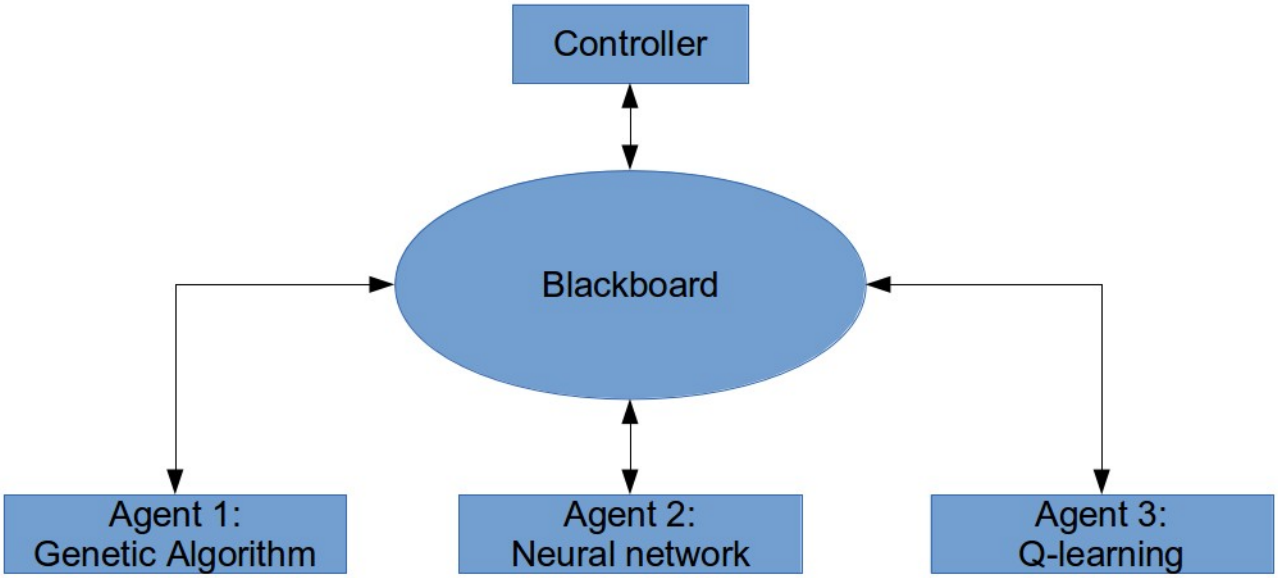


Illustration 3: System wieloagentowy o architekturze „blackboard". W tym przypadku agenci są zaimplementowani na różne sposoby. Nasz algorytm może jednak być użyty jako jądro każdego z agentów w systemie.

Wszyscy agenci współdzielą wiedzę między sobą poprzez tak zwaną „czarną tablicę" (ang. blackboard). Nie jest to już współdzielenie wiedzy takie jak opisaliśmy wcześniej. Tutaj baza wiedzy gromadzi informacje z różnych skrzyżowań i różne skrzyżowania mają do niej dostęp.

Inną formą komunikacji może być wzajemna sieć połączeń między agentami. Polega ona na koncepcji „każdy z każdym". Jej wadą jest duża złożoność i trudności w opracowaniu sposobu interpretacji przekazywanych informacji.

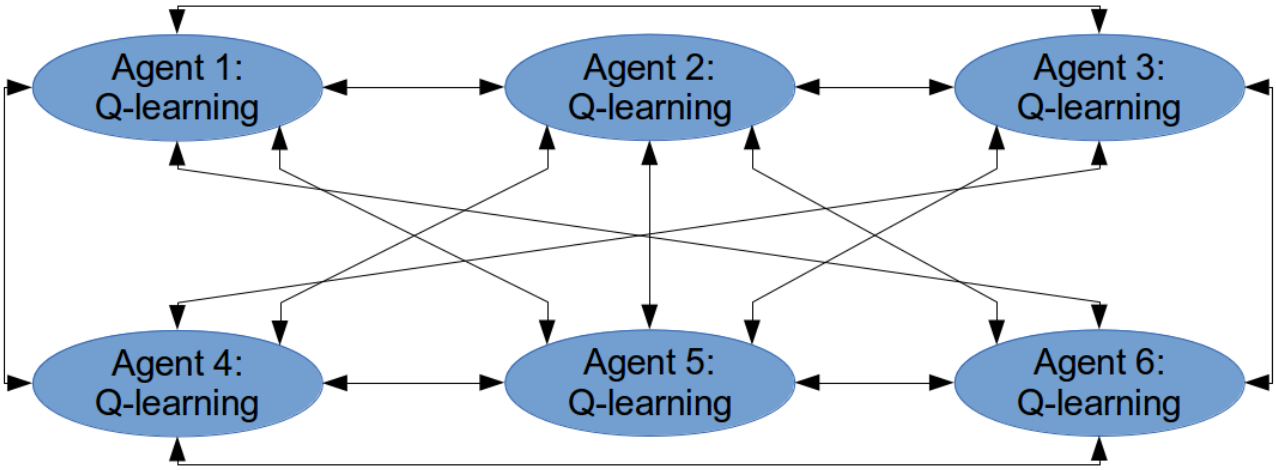


Illustration 4: System wieloagentowy z rozszerzoną komunikacją między agentami.

Istnieje także mniej skomplikowana sieć połączeń. W tym przypadku, agenci komunikują się tylko z wybranymi agentami (najczęściej są to bezpośredni sąsiedzi).

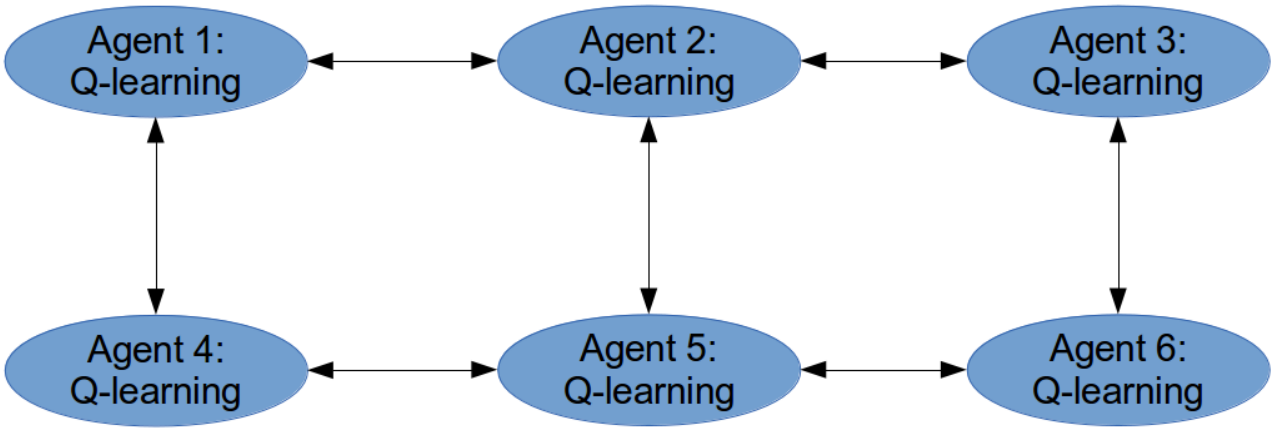


Illustration 5: System wieloagentowy z uproszczoną komunikacją miedzy agentami.

Nasza praca jest zatem jednym z elementów systemu wieloagentowego. Opracowanie algorytmu, który potrafi rozwiązać problem lokalnie, jest kluczowe w implementacji całego systemu wieloagentowego. Mamy nadzieję, że nasze rozwiązanie będzie na tyle efektywne, żeby mogło zostać z powodzeniem zastosowane w jednym z opisanych przez nas systemów.

Zanim zdecydowaliśmy się na zastosowanie Reinforcement Learning w naszej pracy, zrobiliśmy przegląd niektórych rozwiązań. Takie rozeznanie pozwoliło nam na podjęcie decyzji o tym, którą metodę zastosować. W kolejnych podrozdziałach postaramy się przybliżyć najbardziej standardowe podejścia do zarządzania ruchem drogowym. Można o nich powiedzieć, że stanowią bardzo bogate źródło wiedzy na temat metod sterowania. Umieszczamy je w tym miejscu dla zaznajomienia czytelnika z innymi metodami oraz dlatego, że wiele z nich było dla nas inspiracją pomimo tego, że wykorzystuje się w nich odmienne metody.

Ubolewamy nad faktem, że nie jesteśmy w stanie porównać naszego rozwiązania z wynikami innych autorów. Nie jest to możliwe ze względu na złożoność problemu oraz bardzo odmienne środowisko testowe. W celu wykonania miarodajnego porównania, algorytmy powinny być testowane w tych samych warunkach i na tym samym skrzyżowaniu, co nie jest możliwe, gdyż aktualnie istnieje, zbyt duża różnorodność symulatorów oraz skrzyżowań, na których były testowane algorytmy.

2. Algorytmy genetyczne

Algorytmy genetyczne znajdowały zastosowanie w wielu dziedzinach życia. Można o nich przeczytać w wielu pracach i książkach. W celu poszerzenia wiedzy o algorytmach genetycznych polecamy w szczególności książkę „Genetic Algorithm + Data Structures = Evolution Programs” Zbigniewa Michalewicza. Treść tej książki doskonale wprowadza w świat algorytmów ewolucyjnych, pokazuje ich wariacje oraz wady i zalety. Jest te skarbnicą wielu praktycznych przykładów.

Niniejszy rozdział wymaga co najmniej podstawowej wiedzy na temat algorytmów genetycznych, ponieważ skupimy się raczej na pokazaniu problemu i rozwiązania niż na wyjaśnianiu idei algorytmów. Żeby jednak nie zostawić czytelnika, który nie spotkał się jeszcze z tego rodzaju algorytmami, przedstawimy poniżej ogólny, standardowy schemat programu ewolucyjnego.

Człowiek bardzo często szuka rozwiązań swoich problemów w przyrodzie. Przez miliony lat, w procesie ewolucji organizmy wyrobiły w sobie mechanizmy, które pozwoliły im przetrwać. Natomiast te, które się nie przystosowały wyginęły z powodu tak zwanej selekcji naturalnej.

Podobnie działają algorytmy genetyczne. Definiuje się tak zwane chromosomy (lub inaczej osobniki), z których każdy reprezentuje odmienne rozwiązanie rozpatrywanego problemu. W

pierwszym etapie tworzy się losową populację osobników. Następnie ocenia się je według kryteriów dopasowania (funkcja oceny lub funkcja celu). Te, które wykazują się dużym dopasowaniem są poddawane operatorom genetycznym takim jak mutacja oraz krzyżowanie. Pozostałe (z niską wartością funkcji oceny) są usuwane z populacji. W ten sposób otrzymuje się kolejne iteracje, w których osobniki stają się coraz bardzie dopasowane, aż w końcu powstaje super populacja (końcowa), z której wybiera się najlepszego osobnika opisującego rozwiązanie problemu.

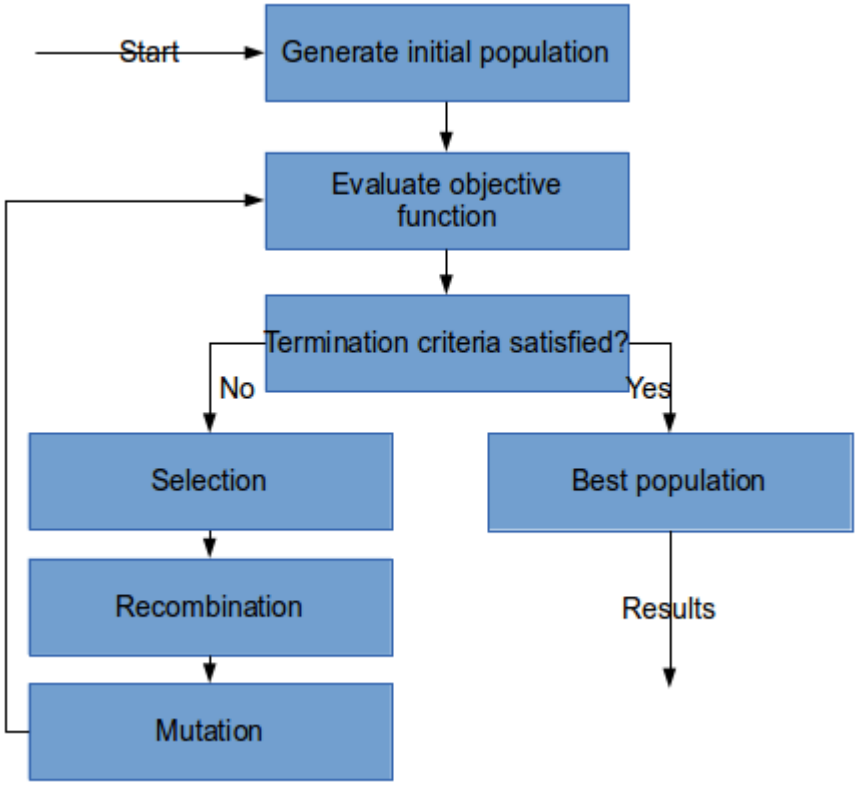


Illustration 6: Schemat działania prostego algorytmu genetycznego.

Wszelkie wady, zalety i subtelności dotyczące algorytmów genetycznych są opisane w [2].

Najczęstszym podejściem do implementacji systemu opartego na algorytmach genetycznych jest optymalizacja ruchu drogowego poprzez manipulowanie długością palenia się zielonych świateł w poszczególnych fazach cyklu. Większość prac obejmuje systemy oparte tylko na pojedynczym skrzyżowaniu (bez dodatkowej synchronizacji między rozwidleniami).

Parametrami, które są brane pod uwagę przy ocenianiu systemu są liczba samochodów na skrzyżowaniu oraz czas, który spędziły oczekując na przejazd przez skrzyżowanie [3].

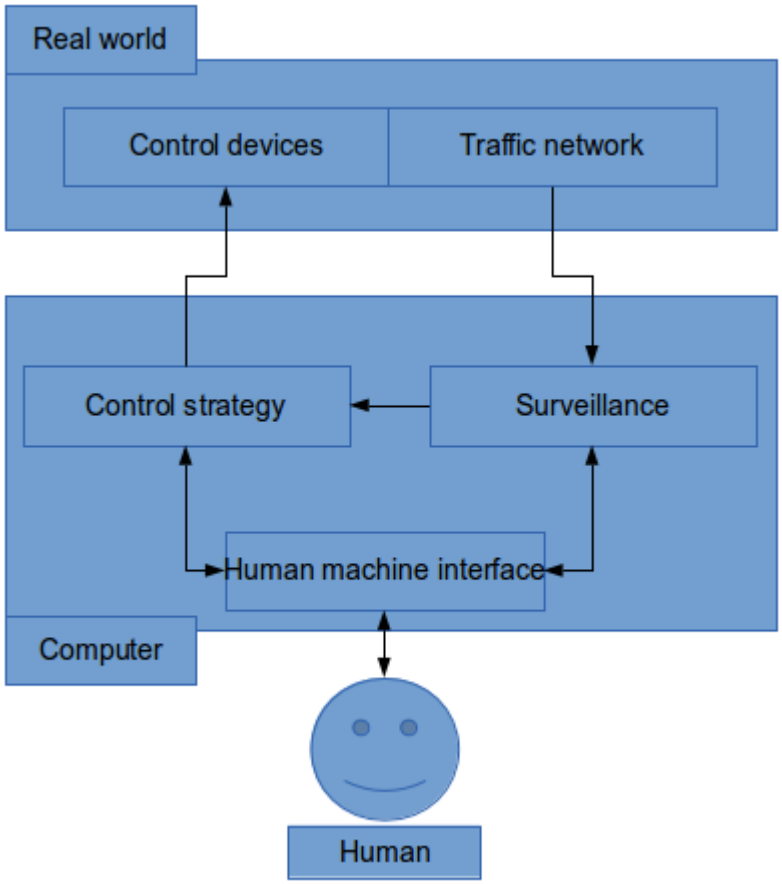


Illustration 7: Model sterowania ruchem drogowym.

Cały system jest konfigurowalny przez człowieka. Ustawia on parametry nadzoru oraz odpowiedni algorytm, który będzie zajmował się określaniem długości palenia się zielonych świateł w danej fazie cyklu. Urządzenia pomiarowe ciągle mierzą sytuację na skrzyżowaniu, to jest liczbę aut, które się na nim znajdują oraz czas, który na nim spędzają. W ten sposób otrzymuje się informacje, które z kolei stają się danymi wejściowymi algorytmu genetycznego (control strategy). Gdy ten wykona potrzebne obliczenia, skomunikuje się z kontrolerem na skrzyżowaniu, który jest odpowiedzialny za ustawienie odpowiednich parametrów w sygnalizatorach.

Zastosowany w [3] algorytm genetyczny opiera się na optymalizacji liniowej funkcji celu: $F = PI_1 + ... + PI_n$, gdzie „n" jest liczbą wszystkich pasów prowadzących do skrzyżowania, natomiast PI Jest tak zwanym indeksem wydajnościowym (performance index). Można go policzyć w następujący sposób:

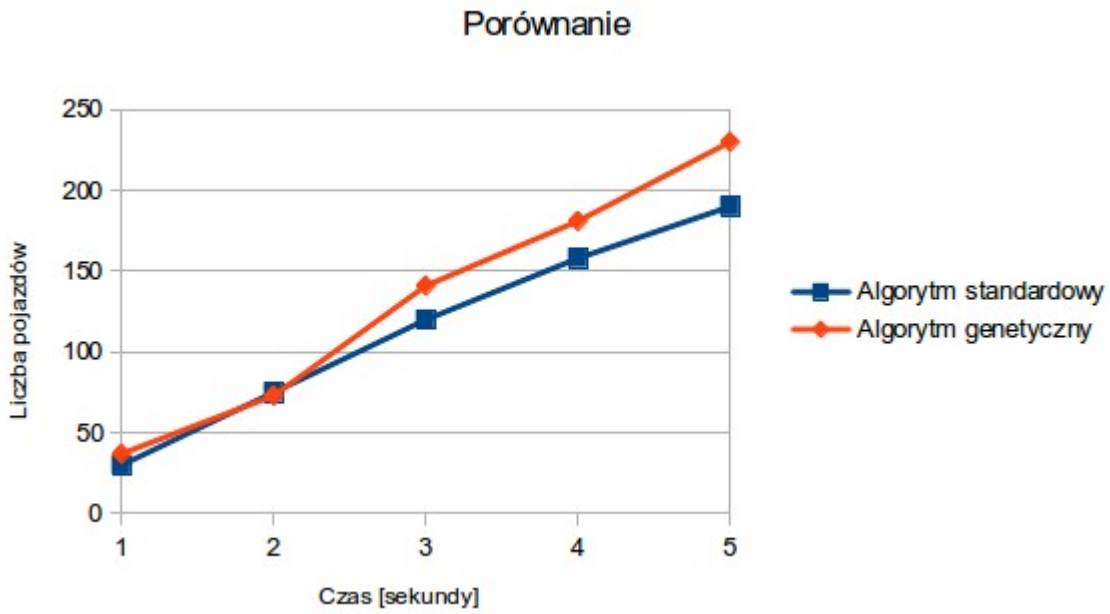
$PI_i = W_i \times (\frac{S_i}{\mathrm{GT}_i})$, gdzie W_i jest wagą danego pasa ruchu, S_i liczbą samochodów na danym pasie ruchu, natomiast GT_i jest czasem palenia się zielonego światła dla tego pasa ruchu. Czas palenia się zielonego światła składa się z tak zwanej podstawy oraz rozszerzenia: $\mathrm{GT}_i = G_{min} + g$, gdzie G_{min} jest minimalnym czasem palenia się zielonego światła (podstawa), a „g" jest rozszerzeniem lub też inaczej wydłużeniem czasu palenia się zielonego światła. Zakłada się dodatkowo, że każdy cykl trwa tyle samo. Z tego założenia wynika następująca własność: $\sum \mathrm{GT}_i = const$.

Algorytm genetyczny otrzymując dane wejściowe przekształca je w serię rozszerzeń („g"), które są podstawą do ustawienia odpowiednio długich faz na urządzeniach sterujących ruchem na skrzyżowaniu.

W omawianej pracy autor osiągnął następujące wyniki w porównaniu do wyników otrzymanych przy zastosowaniu stałego cyklu:

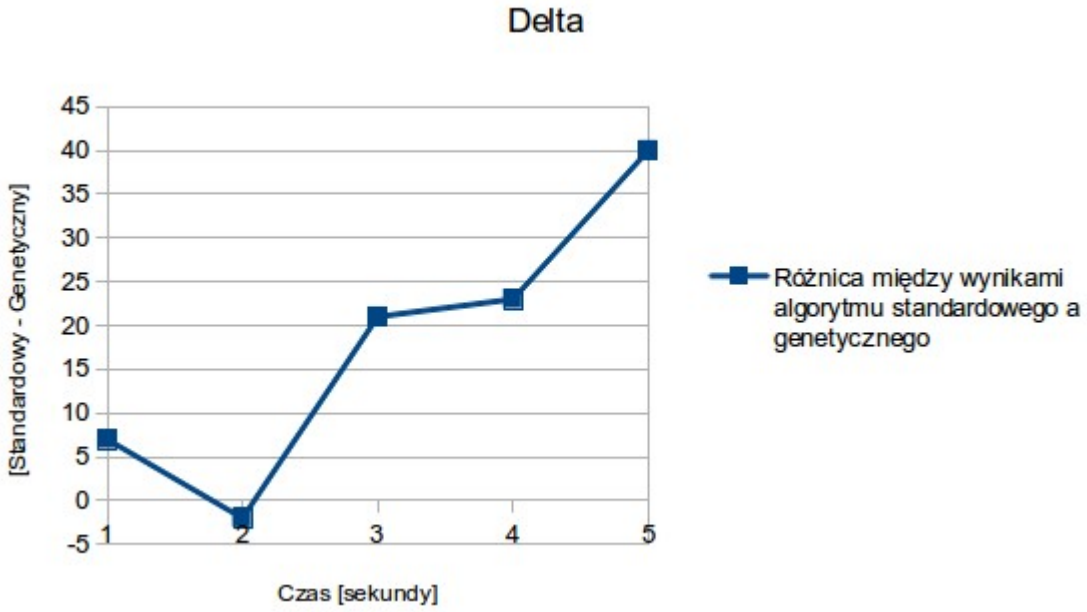
Czas (w minutach)	Standardowy cykl	Algorytm genetyczny
1	30	37
2	75	73
3	120	141
4	158	181
5	190	230
	Suma: 573	Suma: 662

Tabela przedstawia liczbę pojazdów, które przejechały przez skrzyżowanie po upływie danego czasu. Algorytm genetyczny pozwolił na przejechanie o 15% więcej pojazdów niż standardowe rozwiązanie.



Ilustracja 8: Porównanie algorytmu genetycznego ze standardowym.

Wykres pokazuje, że już w drugiej sekundzie algorytm genetyczny dawał lepsze wyniki niż standardowy cykl. Autor niestety nie podaje w swojej pracy dalszych wyników. Biorąc pod uwagę wykres różnic między wynikami algorytmu genetycznego i standardowego i zakładając podobną tendencję wzrostową możemy dojść do wniosku, że dysproporcja między tymi rozwiązaniami powinna się jeszcze zwiększyć w kolejnych sekundach.



Ilustracja 9: Wykres przedstawia różnice w wynikach uzyskanych przez algorytm standardowy i genetyczny.

Powyższy wykres jest rosnący od drugiej sekundy. Oznacza to, że algorytm genetyczny daje coraz lepsze wyniki i jest dosyć stabilny.

Autor [3] opisuje swoje rozwiązanie jako obiecujące i godne dalszej analizy. Jesteśmy jednak sceptycznie nastawieni do zastosowania algorytmu genetycznego w zarządzaniu ruchem drogowym. Nasz sceptycyzm wynika przede wszystkim z trudnościami w określeniu funkcji celu. Bardzo wiele z nich są na tyle proste, że zwykły matematyczny mechanizm mógłby znaleźć rozwiązanie w dużo wydajniejszy sposób. Z drugiej strony, skomplikowane funkcje oceny są niemiarodajne, ponieważ korzystają ze zbyt wielu parametrów. Takie podejście zaciemnia cały problem i sprawia, że algorytm nie działa tak jak powinien. Główną zasadą jest stosowanie funkcji oceny, która skupia się tylko na najważniejszych parametrach abstrahując pozostałe. Należy więc zastanowić się, czy nie warto stosować bardzo skomplikowany mechanizm taki jak algorytmy genetyczne do znalezienia osobnika (rozwiązania), którego jesteśmy w stanie znaleźć korzystając tylko z matematycznych obliczeń.

Dodatkową wadą algorytmów genetycznych jest brak reakcji na zmiany. Algorytm działa zawsze tak samo w oparciu o funkcję celu. Dodatkowo ewolucja populacji zajmuje dosyć dużo czasu, a należy pamiętać, że podczas sterowania ruchem na skrzyżowaniu nie można sobie pozwolić na opóźnienia. Może także zajść sytuacja, gdy podczas obliczeń algorytmu, stan skrzyżowania się zmieni i znalezione rozwiązanie nie będzie już poprawne, a co gorsze może być bardzo dalekie od optymalnego.

Stąd, pomimo uzyskanych przez autora pozytywnych wyników sądzimy, że algorytmy genetyczne są jednym z możliwych sposobów na rozwiązanie problemu sterowania ruchem ulicznym, ale nie są na tyle dobrym sposobem, aby się nimi dalej w tym kontekście zajmować.

3. Sieci neuronowe

Sieci neuronowe opierają się na połączeniach między neuronami. Dobrze zbudowana sieć jest w stanie analogicznie do ludzkiego mózgu uczyć się reakcji na dane zachowania czy sytuacje. Dzięki takiej zdolności może ona przewidzieć jakie będzie nasilenie ruchu w krótkim czasie lub też wybrać odpowiednią akcję, która pozwoli na rozluźnienie sytuacji na skrzyżowaniu.

Budowa sieci jak już wcześniej wspomniałem jest oparta o neurony.

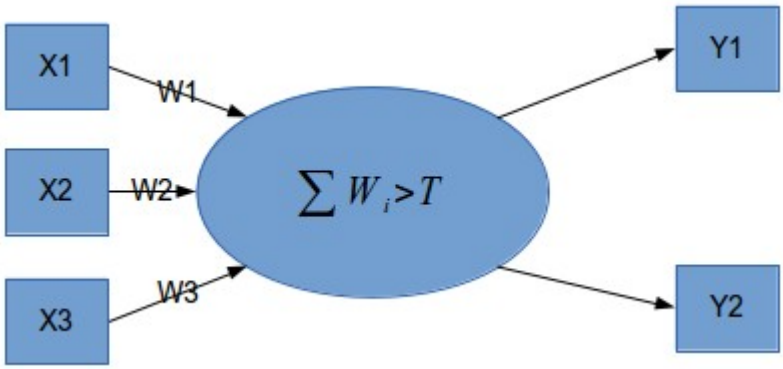


Illustration 10: Budowa prostego neuronu.

Ilustracja 10 przedstawia budowę prostego neuronu. Na wejście przychodzą sygnały z innych neuronów. Każdy z nich ma pewną wagę, która brana jest pod uwagę przy określaniu, czy neuron zostanie aktywowany czy będzie dalej nieaktywny. Aktywacja następuje, gdy zostanie spełniony warunek. Na przykład suma wag będzie większa od pewnej stałej temperatury T. Jeżeli tak się stanie, wyjście Y1 zostanie aktywowane.

Sieci neuronowe charakteryzują się przede wszystkim dużą tolerancją na nieciągłości i zaburzenia. Oznacza to, że dane, które otrzyma sieć na wejściu mogą być zniekształcone, a sieć i tak poradzi sobie z ich analizą. Kolejną zaletą sieci neuronowych jest to, że wystarczy sieć nauczyć raz, aby czerpać później z niej korzyści.

Wiele informacji na temat sieci neuronowych można znaleźć w różnego rodzaju podręcznikach i publikacjach, dlatego nie będę opisywał budowy i działania sieci. Skupię się jednak na aspektach, które są konieczne do zrozumienia dalszej części pracy.

Aby sieć była użyteczna, musi zostać wcześniej nauczona. Istnieje wiele sposobów uczenia sieci:

- Uczenie ze wsteczną propagacją błędów – inaczej uczenie z nadzorem lub nauczycielem. Nauczanie polega na utworzeniu tak zwanych wektorów uczących, które składają się z wektorów wejściowych oraz wyjściowych. Dane w wektorze wejściowym powinny być tak dobrane, aby jak najlepiej reprezentowały badany problem. Z kolei dane wejściowe są używane do sprawdzenia czy sieć zachowała się poprawnie. Sieć może obliczyć błąd jaki popełniła. W tej fazie następuje wsteczna propagacja błędu. Wsteczna, ponieważ odbywa się w przeciwnym kierunku niż miało to miejsce w przypadku wektora wejściowego. Błąd jest propagowany od warstwy wyjściowej do warstwy wejściowej, tak, że w każdej z warstw sieć może wykonać korektę wag w neuronach. W ten sposób, sieć powoli uczy się rozpoznawać dane wejściowe i właściwie na nie reagować. Zbiór wektorów wejściowych nazywany jest ciągiem uczącym. Gdy już wszystkie wektory uczące przejdą przez sieć, kończy się pierwszy etap, czyli tak zwana epoka. Liczony jest wtedy ogólny błąd epoki, a następnie sprawdza się, czy błąd jest wystarczająco niski. Jeżeli nie, powtarza się nauczanie sieci z tym samym ciągiem uczącym aż do uzyskania zadowalającego wyniku [4].
- Sieć Hopfielda – sieć ze sprzężeniem zwrotnym. W takiej sieci istnieje możliwość połączenia wyjścia danego neuronu z wejściem neuronu w tej samej warstwie.
- Uczenie Hebba – inaczej uczenie bez nauczyciela. Opiera się na grupowaniu sygnałów wejściowych w pewne klasy. Taka sieć jest w stanie rozpoznać dane wejściowe i przyporządkować je do jednej z klas podobieństwa.

Autor publikacji naukowej [5] proponuje następujący model systemu sterującego ruchem drogowym w oparciu o sieci neuronowe.

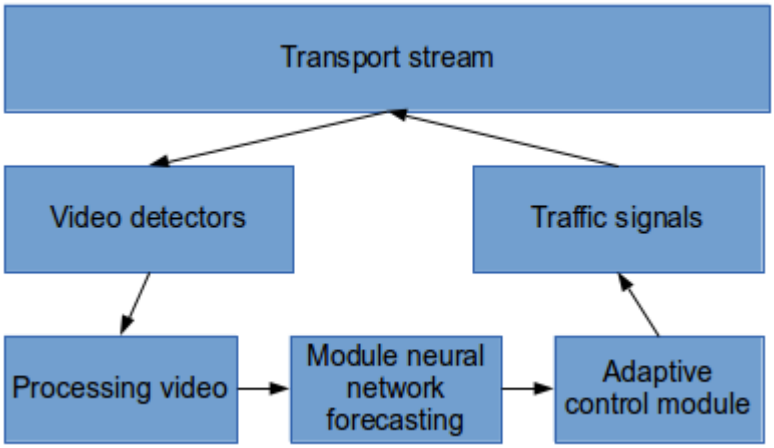


Illustration 11: Model systemu opartego o sieci neuronowe.

Sieć ma za zadanie przewidzieć intensywność ruchu na wszystkich pasach danego skrzyżowania. Na tej podstawie zostanie podjęta decyzja o ustawieniu długości świateł w kolejnych cyklach. Do tego celu użyto wielowarstwowej sieci neuronowej z 400 wejściami, 100 neuronami w warstwie ukrytej i jednym wyjściem. Na wejście składają się dane historyczne, to jest intensywności ruchu na danym pasie w kontekście historycznym. Na ich podstawie sieć przewiduje natężenie ruchu w kolejnej jednostce czasu.

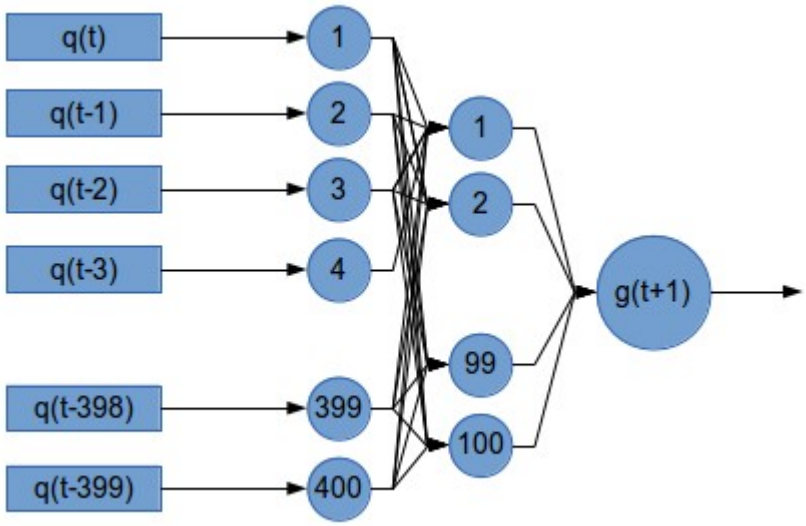


Illustration 12: Perceptron z 400 wejściami, 100 neuronami w warstwie ukrytej oraz jednym wyjściem.

Ponieważ każde ze skrzyżowań zawiera więcej niż jeden pas dojazdowy, każdy musiał zawierać swoją niezależną sieć neuronową. W ten sposób uzyskano wiele sieci, a każda była odpowiedzialna za inny pas opisywanego skrzyżowania.

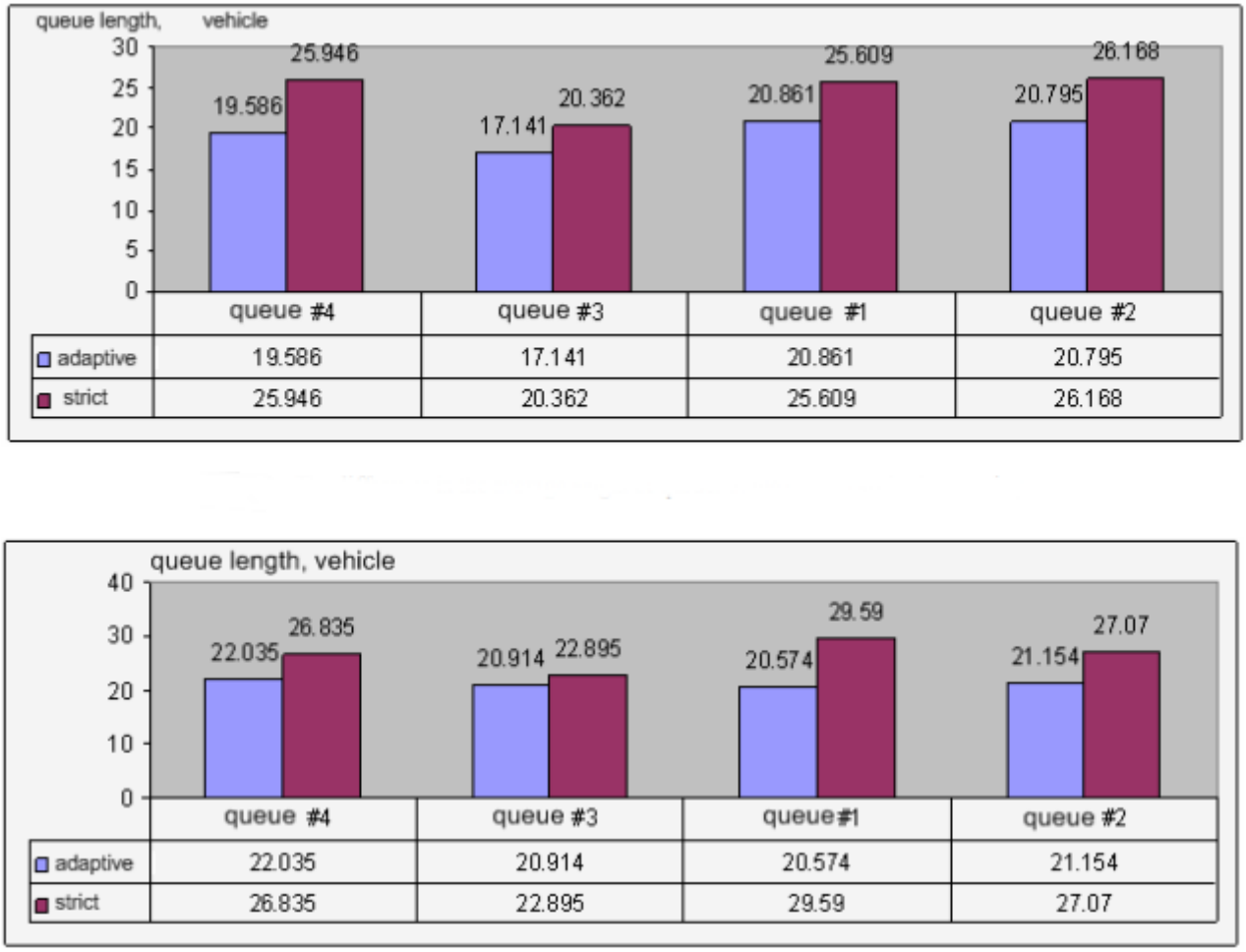
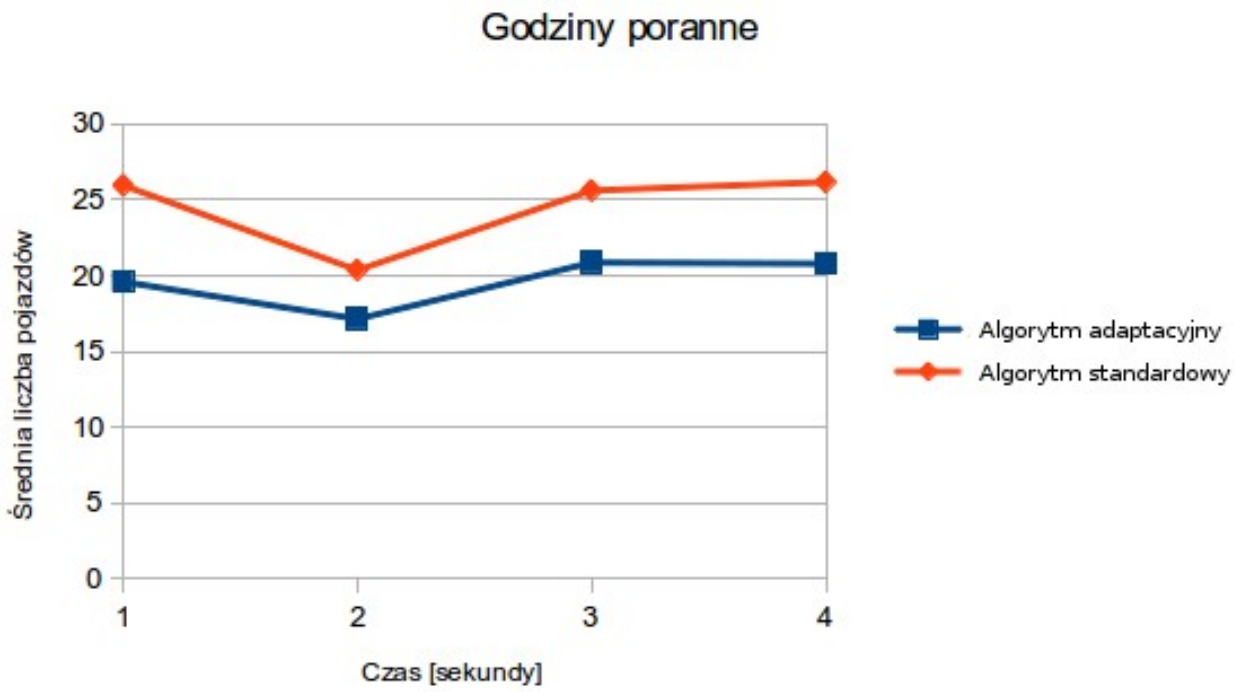


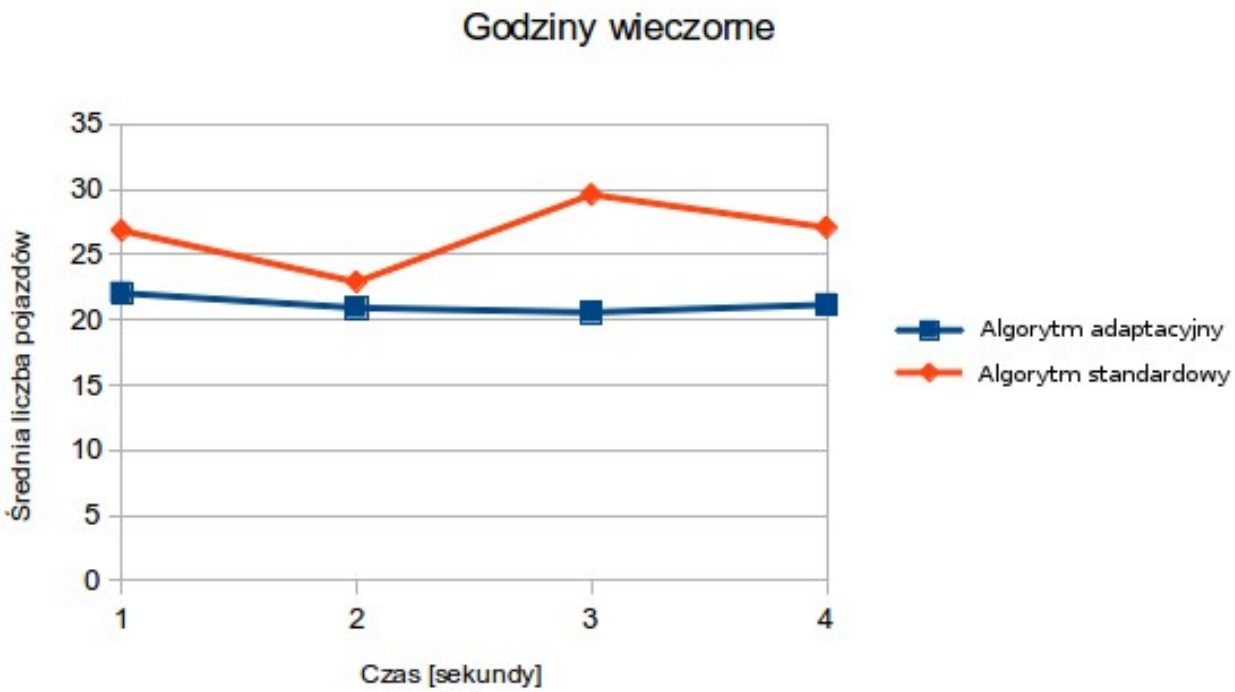
Illustration 13: Wyniki autora, źródło: Artificial neural networks for adaptive management traffic light objects at the intersection, Sergey V.Anfilets, Vasilij N.Shuts

Autor porównał swoje wyniki ze standardowym systemem sterowania ruchem drogowym. Wyniki przedstawił w tabelach. Pierwsza tabela przedstawie rezultaty z badania ruchu ustawionego na tryb poranny, druga natomiast na tryb nocny.



Ilustracja 14: Porównanie wyników uzyskanych przez system adaptacyjny w godzinach porannych.

Powyższy wykres przedstawia średnie długości kolejek na skrzyżowaniu w kolejnych odstępach czasowych. Zarówno poranną porą jak i wieczorną, algorytm adaptacyjny utrzymuje mniejsze kolejki niż algorytm standardowy.



Ilustracja 15: Porównanie wyników uzyskanych przez system adaptacyjny w godzinach wieczornych.

Algorytm oparty na sieci neuronowej, który próbuje przewidzieć jaka będzie intensywność ruchu w kolejnym oknie czasowym okazuje się lepszy od algorytmu standardowego. Sądzimy jednak, że

różnice są zbyt małe a system powinien wykazać się większą wydajnością. Dodatkowo problemem są tutaj zmiany charakteru ruchu. Sieć, która została raz nauczona, będzie korzystała z nabytej wiedzy, która może być nieaktualna wraz ze zmianą parametrów ruchu. Należy zatem w tym przypadku rozważyć sprawdzanie, czy sieć nie potrzebuje modyfikacji posiadanej wiedzy. Może się zdarzyć, że będzie wymagała przeprowadzenia nauczania od początku. Proponowane rozwiązanie jest także kontrowersyjne ze względu na fakt, że przewidywanie intensywności nie rozwiązuje wcale problemu. Należy dodatkowo opracować mechanizm, który na podstawie szacowanej intensywności ustawi odpowiedni czas palenia się zielonego światła na wszystkich fazach cyklu.

Autor [5] uzyskał średnio o 20% lepsze wyniki korzystając z sieci neuronowej niż w przypadku zastosowania algorytmu standardowego. Pomimo zadowalającego wyniku uważamy, że niezależna sieć neuronowa nie jest w stanie osiągnąć większej wydajności. Sieć neuronowa lepiej sprawdza się w rozwiązywaniu problemów rozpoznawania wzorców oraz aproksymowaniu funkcji. Nadaje się także do grupowania danych wejściowych w obrębie pewnej liczby klas. Nie chcemy podważać jej zdolności do rozwiązania problemu sterowania ruchem ulicznym, ale sądzimy, że zastosowanie sieci neuronowej do sterowania ruchem ulicznym bez dodatkowych mechanizmu nie doprowadzi nas do osiągnięcie ponad 30 procentowej przewagi nad algorytmem standardowym.

4. Wnioski z przedstawionych rozwiązań

Istnieje wiele publikacji naukowych, które przedstawiają odmienne spojrzenie na problem sterowania ruchem ulicznym. Nie sposób jednak znaleźć i opisać ich wszystkich. Przytoczone przez nas dwie propozycje [3] i [5] miały na celu przybliżenie czytelnikowi innych podejść, aby mógł bez wcześniejszego zagłębiania się w tematykę porównać inne rozwiązania do naszego.

Każda z prac wnosi coś nowego do zgromadzonej wiedzy na temat inżynierii ruchu drogowego i sterowania ruchem. Znajdujemy wiele interesujących parametrów, rozwiązań, założeń i metod w propozycjach innych autorów.

Naszym celem jest jednak osiągnięcie ponad 30 procentowej przewagi nad standardowymi systemami sterowania ruchem. Jak do tej pory, nie udało nam się znaleźć algorytmu, który osiągnąłby ten cel. Stąd narodził się pomysł połączenia kilku rozwiązań w jedno. Postanowiliśmy zastosować sieć neuronową w połączeniu z Reinforcement Learning. Uzasadnienie tej decyzji przedstawimy jednak w osobnym rozdziale.

III. Reinforcement learning – wprowadzenie

W dzisiejszych czasach komputery rozwiązują wiele problemów ludzkości. Istnieje jednak wiele zagadnień, z którymi maszyny nie mogą sobie poradzić. Nie z powodów ograniczeń sprzętowych (chociaż i takie się zdarzają) ale głównie z powodu braku wiedzy na temat tego, co program powinien robić. Stąd konieczność opracowania metod, które będą się w stanie uczyć się pewnej sytuacji, a następnie rozwiązać zadany problem w oparciu o zdobytą wiedzę.

Stosowano wiele różnych podejść, poczynając od algorytmów genetycznych, poprzez sieci neuronowe, systemy rozmyte czy też programowanie dynamiczne. Każde z tych „rozwiązań” ma swoje wady i zalety oraz odmienne zastosowania. Dopiero wnikliwa analiza problemu pozwala na wybranie odpowiedniej metody oraz dostosowanie jej do środowiska, w którym problem jest rozwiązywany.

Reinforcement Learning jest metodą opracowaną z myślą o usprawnieniu metod takich jak programowanie dynamiczne (bardzo duża złożoność obliczeniowa) oraz nauczanie z nauczycielem (supervised learning).

Problem w nauczaniu z nauczycielem polegał na tym, że musieliśmy zadawać pytania, na które znaliśmy odpowiedzi. Na podstawie tych pytań system uczył się odpowiadać na te, które jeszcze nigdy nie były mu zadane. Niestety, czasami bywa tak, że nie znamy poprawnych odpowiedzi na zadane pytania. Wtedy nauczanie z nauczycielem traci nie tyle sens, co w ogóle rację bytu.

Stąd potrzeba opracowania RF (Reinforcement Learning), które jest połączeniem programowania dynamicznego oraz nauczania z nauczycielem. Ponieważ czerpie z tych metod ich wszystkie zalety, jest w stanie rozwiązywać bardziej skomplikowane zadania i poszukiwać odpowiedzi na trudniejsze pytania.

Aby lepiej zrozumieć na czym polega nauczanie ze wspomaganiem przytoczymy prosty przykład. Weźmy znaną kiedyś grę „ciepło-zimno”. Pierwsza osoba (ukrywający) chowa pewną rzecz w pomieszczeniu. Druga osoba (szukający) ma za zadanie odnaleźć tą rzecz w oparciu o podpowiedzi chowającego. Informacja „ciepło” oznacza, że szukający zbliża się do ukrytej rzeczy, natomiast informacja „zimno” mówi szukającej osobie, że ukryta rzecz znajduje się w innym miejscu. Strategia szukającego polega na chodzeniu po pokoju i odbieraniu sygnałów od osoby, która schowała przedmiot. Na tej podstawie uczy się i odnajduje schowaną rzecz. Jeżeli zrobi krok, który oddala go od przedmiotu, zostaje „ukarany” komendą „zimno”. Po znalezieniu rzeczy, osoba szukająca jest w stanie już bez podpowiedzi znaleźć jeszcze raz ten sam przedmiot w oparciu o swoje doświadczenie. Nauczyła się bowiem akcji (kroków), które powinna wykonać w celu dojścia do ukrytego przedmiotu (mówimy tutaj o przedmiocie, który jest ukryty ciągle w tym samym miejscu).

Przykład chociaż banalny, bardzo dobrze ukazuje specyfikę nauczania ze wspomaganiem. W literaturze przytaczane są także przykłady takie jak nauka jazdy na rowerze, gra w backgammon czy też problem „Car on the hill” [6], którego opis znajduje się w wielu książkach o nauczaniu maszynowym.

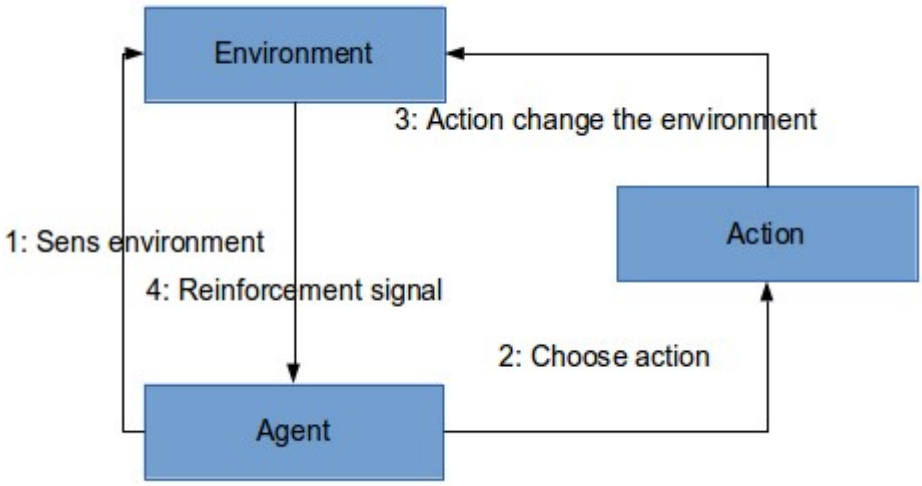


Illustration 16: Podstawowa zasada działania RL. Agent (jednostka, która się uczy) odbiera świat za pomocą sensorów. Na podstawie zgromadzonych danych wybiera jedną z możliwych akcji, które wpływają na środowisko, w którym agent funkcjonuje. Następnie Agent odbiera nagrodę lub karę za wykonaną akcję w zależności od tego, czy akcja przyniosła planowany skutek.

Nauczanie ze wspomaganiem składa się z kilku ważnych elementów, z którymi czytelnik powinien się zapoznać w celu zrozumienia dalszych rozdziałów.

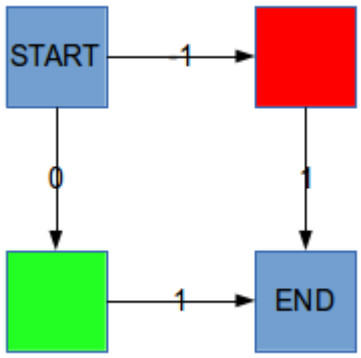
IV. Reinforcement Learning – części składowe

1. Środowisko

Ten element jest otoczeniem agenta. To właśnie dzięki niemu agent będzie zdobywał wiedzę potrzebną do rozwiązania problemu. Aby to było możliwe środowisko musi być co najmniej częściowo obserwowalne. Oznacza to, że za pomocą sensorów, agent jest w stanie pobrać ze środowiska potrzebne mu informacje. Od dokładności tych informacji zależy późniejszy wybór akcji. Dlatego tak ważne jest, aby agent mógł pobierać dane prawdziwe i niezniekształcone.

2. The Reinforcement Function – funkcja wzmocnienia

Funkcja wzmocnienia odgrywa bardzo ważną rolę w RL. Dzięki niej zdefiniowany jest cel nauczania. Najczęściej jest to funkcja dwóch zmiennych, stanu środowiska oraz akcji, która danej parze przypisuje pewną liczbę rzeczywistą. Jest to tak zwane wzmocnienie (lub czasami osłabienie). Na podstawie wartości tej funkcji agent będzie w stanie stwierdzić, czy wykonana przez niego akcja przybliża go do celu czy też nie. RL opiera się właśnie na maksymalizowaniu sumy wszystkich otrzymanych wzmocnień poczynając od stanu początkowego do stanu określanego jako końcowy.



Ilustracja 17: Przykład prostego zadania polegającego na przejściu ze stanu START do stanu END.

Rozważmy zadanie, które jest schematycznie przedstawione na rysunku 17. Możemy zdefiniować 4 stany:

1. Stan początkowy – START
2. Stan czerwony – stan niepożądany, który należy unikać
3. Stan zielony – stan neutralny
4. Stan końcowy – END

Czarne strzałki symbolizują akcje, które możemy wykonywać w danym stanie. Wartości wypisane na strzałkach to nic innego jak wartości funkcji wzmocnienia. Algorytm otrzymuje wzmocnienie, gdy osiąga stan końcowy (+1), nie otrzymuje wzmocnienia, gdy przechodzi do stanu zielonego (0) oraz otrzymuje karę za przejście do stanu czerwonego (-1).

Rozpatrzmy możliwe przypadki. Algorytm zaczyna ze stanu START, z którego może wykonać akcję IDŹ DO STANU CZERWONEGO lub akcję IDŹ DO STANU ZIELONEGO. Jeżeli funkcja wzmocnienia zostanie zdefiniowana jako funkcja dwóch zmiennych, stanu i akcji, to jej tablica wartości będzie się przedstawiała następująco:

States/Actions	Go To Red	Go To Green	Go To End
START	-1	0	x
RED	x	x	1
GREEN	x	x	1
END	x	x	1

Ilustracja 18: Wartości funkcji wzmocnienia. X oznacza, że dana akcja nie jest dostępna dla danego stanu.

W celu pokonania trasy algorytm ma do wyboru dwie niezależne drogi. Pierwsza, która łączy się z przejściem przez zielone pole charakteryzuje się sumą funkcji wartości (odczytanych z tabeli) równą 1, natomiast druga, prowadząca przez pole czerwone będzie miała wartość 0. Stąd algorytm będzie faworyzował ścieżkę pierwszą, ze względu na jej lepszą ocenę (1 jest większe od 0).

Funkcje wzmocnienia mogą być definiowane w różnych sposób (w tym także w postaci tabelarycznej). Istnieją jednak trzy główne podejścia, o których warto wiedzieć podczas pracy nad RL.

Pure Delayed Reward (opóźniona nagroda) and Avoidence Problem (unikanie problemu). Funkcja taka zwraca zera w przypadku, gdy podjęta akcja nie prowadzi do stanu końcowego oraz pewną niezerową wartość w przeciwnym przypadku. Znak tej wartości jest uzależniony od tego, czy osiągnięciu tego stanu towarzyszy sukces czy porażka.

Dla przykładu rozważmy grę „kółko i krzyżyk”. Stan może być tutaj zdefiniowany jako macierz 9 elementowa zawierająca „X” oraz „O”. Załóżmy, że agent gra „X”. Za każdego postawionego „X”, który nie prowadzi do stanu końcowego (stan, gdy wszystkie pola są wypełnione) agent otrzymuje zerowe wzmocnienie. W przypadku, gdy postawienie „X” wyczerpuje wszystkie możliwości, agent może dostać wzmocnienie +1, w przypadku, gdy gra została wygrana oraz -1, w przypadku porażki. W rezultacie po wielu grach, gracz wie jak grać, aby maksymalizować sumę wszystkich nagród, to jest wykonywać ruchy, które doprowadzą go do zwycięstwa. W takim podejściu, w którym za wszystkie ruchu poza końcowym agent dostaje zerowe wzmocnienie, nie ma znaczenia czas gry, ważny jest tylko efekt końcowy.

Kolejnym podejściem do funkcji wzmocnienia jest tak zwana metoda „Minimum Time to Goal”, która opiera się na jak najszybszym wykonaniu zadania. Dla przykładu rozważymy tutaj wcześniej wspomniany problem „Car on the Hill”.

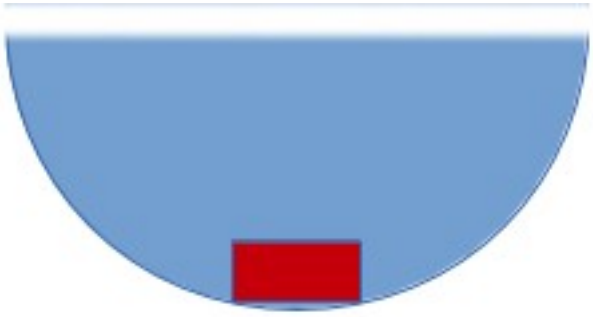


Illustration 19: Samochód znajduje się na wzniesieniu. Problem polega na wyjechaniu z delty, przy założeniu, że pojazd jest zbyt słaby, aby opuścić dolinę za pierwszym razem. Jedynym sposobem na wydostanie się jest umiejętne rozpędzanie się na przeciwległych podjazdach.

Określamy stan jako prędkość pojazdu oraz jego pozycję. W każdym ze stanów możliwe są trzy

akcje. Pchanie pojazdu do przodu, do tyłu lub w ogóle. Pojazd otrzymuje negatywne wzmocnienie -1 za każdą akcję, która nie prowadzi go do stanu końcowego. Za wyjechanie z dołu (stan ostateczny) agent otrzymuje zerowe wzmocnienie. Ponieważ agent chce maksymalizować sumę wzmocnień, będzie się starał wyjechać jak najszybciej. Proszę zwrócić uwagę na fakt, że każda akcja, która nie prowadzi do stanu końcowego zmniejsza tę sumę o jeden. W tym przypadku ważny jest więc czas, stąd nazwa „Minimum Time to Goal".

Trzecim podejściem do definiowania funkcji wzmocnienia są tak zwane „gry". W takim przypadku okazuje się, że nie zawsze zależy nam na maksymalizacji sum. Czasami warto jest szukać minimalnej sumy. Takie podejście sprawdza się w niektórych grach, w których gracze mają sprzeczne cele.

Rozważmy samolot oraz rakietę, która została wystrzelona w celu zniszczenia samolotu. Samolot będzie miał za zadanie maksymalizować dystans od rakiety, podczas gdy rakieta będzie chciała, aby ten dystans był coraz mniejszy. Stąd sprzeczne cele i zupełnie inne podejście do wzmacniania w RL.

3. Value Function – funkcja wartości

Funkcja wartości jest istotą nauczania w RL. To na jej podstawie agent uczy się, które akcje podejmować w celu uzyskania stanu końcowego. Funkcja wartości jest powiązana z dwoma terminami, które trzeba zrozumieć przed kontynuacją czytania tej pracy.

Policy – strategia wyboru akcji w danym stanie. Strategia jest więc mapowaniem stanu w akcję. Może ona się opierać na przykład na maksymalizowaniu sumy wszystkich otrzymywanych wzmocnień między stanami skrajnymi (początkowym i końcowym).

Value – suma wzmocnień otrzymanych poczynając od stanu początkowego do stanu końcowego z uwzględnieniem danej strategi (policy).

Funkcja wartości jest więc rodzajem pamięci agenta, jego doświadczeniem, które zmienia się w trakcie nauki. To właśnie na jej podstawie będą podejmowane decyzje już nauczonego agenta. Jest wiele sposobów definiowania funkcji wartości. Może to być dowolny aproksymator funkcji lub też zwykła tabela. Jednym z problemów RL jest określenie sposobu na znalezienie optymalnej postaci funkcji wartości.

4. Szacowanie funkcji wartości

Istnieją dwie główne zasady przy definiowaniu metody aproksymacji funkcji wartości. Pierwsza zasada opiera się na założeniu, że jeżeli akcja w danej sytuacji prowadzi do tragedii, system nauczy się, że nie należy jej wykonywać w danym stanie. Druga zasada dotyczy stanów. System uczy się, że nie należy doprowadzać do stanów, które prowadzą do porażki.

4.1. Szacowanie funkcji wartości w oparciu o programowanie dynamiczne

Jednym ze sposobów na szacowanie funkcji wartości jest programowanie dynamiczne. Zanim przejdziemy do opisu tego podejścia należy wprowadzić potrzebną notację:

- $V^*(x_t) - wartość\ optymalna\ funkcji\ dla\ wektora\ stanu\ x_t$
- $V(x_t) - wartość\ szacowana\ funkcji\ dla\ wektora\ stanu\ x_t$
- $\gamma - współczynnik\ nauczania\ z\ przedziału [0,1]$

W praktyce, na samym początku uczenia, wartości szacowane są wybierane losowo i nie odbiegają w znacznym stopniu od wartości optymalnej. Oznacza to, że szacowanie charakteryzuje się pewnym błędem $e(x_t)$:

$$V(x_t)=e(x_t)+V^*(x_t) \quad (1)$$

Taka sama sytuacja będzie miała miejsce w kolejnej iteracji:

$$V(x_{t+1})=e(x_{t+1})+V^*(x_{t+1}) \quad (2)$$

Wartość $V(x_t)$ dla optymalnej strategii jest sumą wzmocnień zaczynając od stanu opisanego przez wektor x_t aż do stanu końcowego. Na podstawie tej definicji można określić zależność między wartościami kolejnych stanów. Ta zależność jest wyrażona we wzorze Bellmana:

$$V^*(x_t)=r(x_t)+\gamma V^*(x_{t+1}) \quad (3)$$

We wzorze (3) pojawia się wartość funkcji nagrody (wzmocnienia) $r(x_t)$.

Taka sama sytuacja zachodzi dla wartości szacowanych:

$$V(x_t)=r(x_t)+\gamma V(x_{t+1}) \quad (4)$$

Po podstawieniu do prawej strony równania (4) wartości z równania (1) możemy otrzymać następującą zależność:

$$V(x_t)=r(x_t)+\gamma V(x_{t+1})$$

$$e(x_t)+V^*(x_t)=r(x_t)+\gamma V(x_{t+1})$$

$$e(x_t)+V^*(x_t)=r(x_t)+\gamma V(x_{t+1})$$

$$e(x_t)+V^*(x_t)=r(x_t)+\gamma(e(x_{t+1})+V^*(x_{t+1}))$$

Stosujemy wzór (3):

$$e(x_t)+r(x_t)+\gamma V^*(x_{t+1})=r(x_t)+\gamma(e(x_{t+1})+V^*(x_{t+1}))$$

Po skróceniu otrzymujemy:

$$e(x_t)=\gamma e(x_{t+1})$$

Z powyższego równania wynika bardzo ważny fakt. Zakładając, że błąd przy przejściu do ostatniego stanu jest równy 0, wszystkie poprzednie błędy także będą równe zeru. Znajdziemy zatem optymalną postać funkcji wartości.

4.2. Value iteration

Iteracja wartości opiera się na krokowym zmniejszaniu błędu między szacowaną wartością funkcji a jej optymalną wartością. Aby osiągać coraz lepsze przybliżenia w każdej kolejnej iteracji stosuje się zmodyfikowany wzór Bellmana:

$$e(x_t)=\max_a(r(x_t,a)+\gamma V(x_{t+1}))-V(x_t)$$

Zgodnie z poprzednimi wyprowadzeniami błąd jest różnicą między szacowaną wartością a wartością optymalną i dla nieskończenie wielu iteracji dąży do zera.

Takie podejście jest jednak bardzo złożone obliczeniowo. Wymaga wykonania każdej z akcji, a następnie sprawdzenie wartości stanu, do którego ona doprowadziła. Stąd też stosuje się bardzo często funkcję Q.

4.3. Q-Learning

Q-learning zastępuje funkcję wartości tak zwaną funkcją Q, której wartość zależy od akcji oraz stanu. Funkcja Q zwraca sumę wszystkich wzmocnień otrzymanych po wykonaniu danej akcji przy stosowaniu zadanej strategii.

Wróćmy zatem do przykładu rozpoczętego w poprzednim podrozdziale. Określiliśmy w nim wartości funkcji wzmocnienia w postaci tabelarycznej. Intuicyjnie obliczyliśmy także, która droga będzie najlepsza przy przejściu ze stanu początkowego do stanu końcowego. Teraz dojdziemy do tego samego rezultatu, wykorzystując funkcję wartości, zwaną także funkcją Q. Funkcja ta przedstawia aktualny poziom uczenia algorytmu i jest funkcją zależną od stanu oraz akcji. Akcja jest natomiast wybierana zgodnie ze strategią (policy). W naszym przykładzie strategia będzie się opierała na wybieraniu akcji, dla której wartość funkcji Q jest największa. Funkcja wartości ma postać Q(stan,akcja) i może być zapisana dla dyskretnych problemów w postaci tabeli, podobnie jak funkcja wzmocnienia.

States/Actions	Go To Red	Go To Green	Go To End
START	0	0	0
RED	0	0	0
GREEN	0	0	0
END	0	0	0

Ilustracja 20: Funkcja wartości dla dyskretnych problemów przedstawiona w postaci tabeli.

Bardzo często inicjuje się tabelę zerami. Nie jest to zasada i jeżeli zachodzi taka konieczność lub wymóg można wypełnić ją innymi wartościami.

Funkcja Q zgodnie z teorią powinna być zbieżna do funkcji Q*, która zwraca optymalne rozwiązanie. Aby Q była zbieżna do Q*, należy wybrać odpowiednią metodę jej aktualizacji. Na potrzeby przykładu zdefiniujemy ją jako:

$$Q(stan_i, akcja_i) = R(stan_i, akcja_i) + \max_{akcja}(Q(stan_{i+1}, akcja))$$

Dla wygody definiujemy skróty:

- S_p - stan początkowy
- S_k - stan końcowy
- S_c - stan czerwony
- S_z - stan zielony
- A_c - akcja przejścia do stanu czerwonego
- A_z - akcja przejścia do stanu zielonego
- A_k - akcja przejścia do stanu końcowego

Pierwszy epizod:

1. Funkcja wartości jest inicjalizowana zerami
2. Algorytm jest w stanie początkowym
3. Algorytm może wykonać akcję przejścia do stanu czerwonego lub zielonego
4. Algorytm wybiera losowo akcję przejścia do stanu czerwonego (dla wartości, które są takie same wybiera się akcje w sposób losowy)
5. Następuje aktualizacja funkcji Q zgodnie z metodą aktualizacji.
6. $$Q(S_p, A_c) = R(S_p, A_c) + \max_{akcja}(Q(S_c, akcja)) = -1 + 0 = -1$$
7. Funkcja wartości została zaktualizowana

States/Actions	A_c	A_z	A_k
S_p	-1	0	0
S_c	0	0	0
S_z	0	0	0
S_k	0	0	0

Ilustracja 21: Aktualna tabela wartości funkcji Q.

8. Algorytm jest w stanie S_c
9. Jedyna możliwa do podjęcia akcja to A_k
10. Algorytm wykonuje akcję A_k
11. Aktualizacja funkcji Q
12. $Q(S_c, A_k) = R(S_c, A_k) + \max_{akcja}(Q(S_k, akcja)) = 1 + 0 = 1$
13. Algorytm osiągnął stan końcowy
14. Zakończenie pierwszego epizodu z następującą tabelą wartości:

States/Actions	A_c	A_z	A_k
S_p	-1	0	0
S_c	0	0	1
S_z	0	0	0
S_k	0	0	0

Ilustracja 22: Tabela wartości Q po pierwszym epizodzie.

Drugi epizod:

1. Algorytm jest w stanie początkowym
2. Algorytm może wykonać akcję przejścia do stanu czerwonego lub zielonego
3. Algorytm wybiera akcję przejścia do stanu zielonego, ponieważ:

$Q(S_p, A_c) = -1 < Q(S_p, A_z) = 0$

4. Następuje aktualizacja funkcji Q zgodnie z metodą aktualizacji.
5. $Q(S_p, A_z) = R(S_p, A_z) + \max_{akcja}(Q(S_z, akcja)) = 0 + 0 = 0$
6. Funkcja wartości nie ulega zmianie
7. Algorytm jest w stanie S_z
8. Jedyna możliwa do podjęcia akcja to A_k
9. Algorytm wykonuje akcję A_k
10. Aktualizacja funkcji Q
11. $Q(S_z, A_k) = R(S_z, A_k) + \max_{akcja}(Q(S_k, akcja)) = 1 + 0 = 1$

12. Algorytm osiągnął stan końcowy
13. Zakończenie pierwszego epizodu z następującą tabelą wartości:

States/Actions	A_c	A_z	A_k
S_p	-1	0	0
S_c	0	0	1
S_z	0	0	1
S_k	0	0	0

Ilustracja 23: Tabela wartości funkcji Q po drugim epizodzie.

Etap uczenia (aktualizacji wartości funkcji Q) uznajemy za zakończony. Ponieważ przykład jest elementarny wystarczą dwa epizody do nauczenia algorytmu.

Następnie można już korzystać z nauczonego algorytmu korzystając na przykład ze strategii zachłannej, która preferuje jak najwyższe wartości.

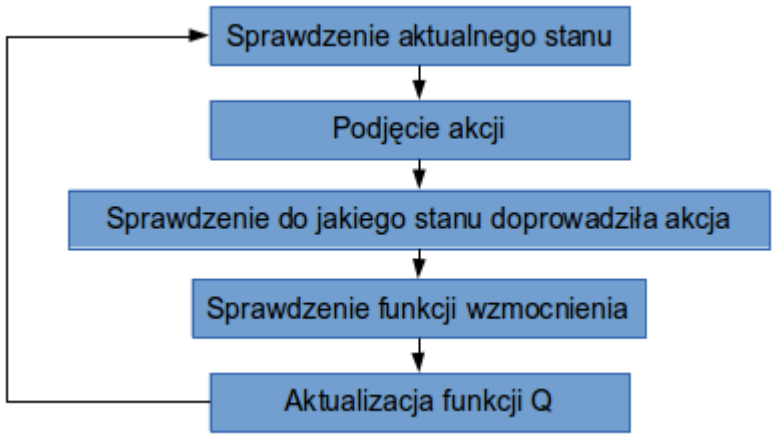
Zaczynając w stanie początkowym, wybieramy akcję, dla które funkcja Q przyjmuje najwyższą wartość. Z tabeli wynika, że jest to akcja przejścia do stanu zielonego (wartość 0). W stanie zielonym przechodzimy do stany końcowego, ponieważ jest to jedyna możliwa akcja.

Algorytm nauczył się jak przejść ze stanu początkowego omijając przy tym stan czerwony zgodnie z założeniami zadania.

Wzór Bellmana dla funkcji Q jest następujący:

$$Q(x_t, a_t) = r(x_t, a_t) + \gamma \max_{a_{t+1}} (Q(x_{t+1}, a_{t+1}))$$

W porównaniu do Value Iteration Q-learning nie wymaga wykonania każdej akcji, a następnie obliczenia szacowanej wartości kolejnego stanu. Jedyne co jest potrzebne, to odczytanie największej wartości funkcji Q dla dowolnej akcji i ustalonym stanie w czasie (t+1).



Rysunek 3: Schemat działania standardowego algorytmu opartego na Q-Learning. W naszym rozwiązaniu zostanie on zmodyfikowany i dostosowany do specyfiki zadania.

RL składa się z dwóch niezależnych etapów: etapu uczenia i etapu eksploatacji. Rysunek 3 przedstawia uproszczony proces uczenia, w którym algorytm modyfikuje wartości funkcji Q. Eksploatacja algorytmu jest oparta na wyborze strategii eksploatacji i na jej podstawie podejmowaniu decyzji, która akcja powinna zostać podjęta w danym stanie.

V. Założenia i cel pracy

5. Założenia

Zanim przejdziemy do opisu konkretnego rozwiązania chcielibyśmy przedstawić założenia, na których będziemy się opierali we wszystkich etapach pracy:

1. System działa niezależnie i optymalizuje pracę pojedynczego skrzyżowania.
2. Testy algorytmów są przeprowadzane w takich samych warunkach.
3. Sprawdzenie działania pojedynczego algorytmu może opierać się na elementach losowych ruchu.
4. Pojazdy uprzywilejowane mają zawsze pierwszeństwo i system powinien pozwolić im na bezpieczny przejazd.

6. Cel

Główny celem naszej pracy jest opracowanie systemu sterującego ruchem drogowym na pojedynczym skrzyżowaniu oraz implementacja symulatora umożliwiającego testy wcześniej wspomnianego systemu. Algorytm powinien przewyższać standardowy algorytm o co najmniej 30 procent.

VI. Rozwiązanie oparte o nauczanie ze wspomaganiem z wykorzystaniem sieci neuronowej

Głównym celem wprowadzenia sygnalizacji świetlnej było zwiększenie bezpieczeństwa na drogach. Kierowca, który dostaje prosty przekaz w postaci światła zielonego lub czerwonego nie musi kontrolować całości skrzyżowania. Dzięki takiemu rozwiązaniu unika się wielu zderzeń i wypadków. Pojazd jest wpuszczony na skrzyżowanie, jeżeli jest bezpiecznie lub pozostaje przed skrzyżowaniem, oczekując na oczyszczenie skrzyżowania.

Oczywiście dalej głównym celem sygnalizacji świetlnej jest zwiększenie bezpieczeństwa na drodze. Niestety, w czasach, gdy liczba samochodów wzrasta w bardzo szybkim tempie, a infrastruktura drogowa się nie zmienia, ważne jest, aby uzyskać założony cel w jak najbardziej optymalny sposób. Chodzi nam tutaj głównie i takie sterowanie ruchem ulicznym, które zapewnia bezpieczeństwo, ale także pozwala na szybki przejazd przez skrzyżowanie. Dlaczego tak ważne jest zwiększenie przepustowości skrzyżowań zostało wyjaśnione w rozdziale wprowadzającym.

Często jest tak, że pojedyncze skrzyżowanie nie może sobie poradzić z nasilającym się ruchem. Wtedy konieczne jest spojrzenie makroskopowe na całą sytuację i próba rozwiązania problemu globalnie w oparciu o informacje gromadzone przez różne skrzyżowania. W tym celu implementuje się tak zwane systemy wieloagentowe, w których rolę agentów przejmują pojedyncze skrzyżowania. Jak już zostało to wspomniane wcześniej istnieje wiele realizacji takich systemów. Zwykle agent działa autonomicznie do czasu, aż system stwierdzi, że dane skrzyżowanie nie jest w stanie poradzić sobie samo z nasilającym się ruchem.

W tej pracy nie będziemy zajmowali się systemami wieloagentowym, ponieważ naszym głównym założeniem jest opracowanie algorytmu, który będzie w stanie zoptymalizować ruch na pojedynczym skrzyżowaniu oraz poinformować o tym, że sytuacja przekracza możliwości danego skrzyżowania. Praca ma stanowić implementację pojedynczego agenta, który potrafi radzić sobie lokalnie ze sterowaniem pojazdami na skrzyżowaniu, a jednocześnie będzie podstawą do badań nad systemem wieloagentowym z użyciem agentów opartych na algorytmie Q-Learning.

Wprowadzenie systemu wieloagentowego oraz wielu skrzyżowań mogłoby doprowadzić do zaciemnienia ważnych aspektów oraz przekłamać niektóre wyniki. Ważne jest, żeby opracować i

udokumentować działanie algorytmu na pojedynczym skrzyżowaniu, a dopiero później próbować zająć się wieloma skrzyżowaniami.

Aby osiągnąć nasz cel będziemy wykorzystywali zmodyfikowany algorytm Q-Learning, który dodatkowo zostanie rozszerzony o sieć neuronową.

Nie zajęliśmy się natomiast algorytmami genetycznymi ze względu na trudności z określeniem funkcji celu. Jesteśmy nastawieni do nich sceptycznie ze względu na fakt, że dobrze napisany algorytm oparty tylko i wyłącznie na prostych obliczeniach matematycznych jest w stanie działać bardzo podobnie jak algorytm genetyczny, a co więcej będzie to robił szybciej. Takie wnioski pojawiły się podczas naszej pracy inżynierskiej, gdy stosowaliśmy do zarządzania ruchem drogowym właśnie algorytmy genetyczne. Wszystko zależy od zdefiniowania funkcji celu, która jest bardzo ciężka do określenia. Stąd lepsze wydają się metody, które mogą się jej nauczyć, lub chociaż mogą posłużyć się jej aproksymacją znalezioną podczas procesu nauki.

Każde ze skrzyżowań posiada zestaw urządzeń pomiarowych, za pomocą których mierzony jest ruch pojazdów. Urządzenia te zliczają liczbę samochodów oczekujących na danym skrzyżowaniu oraz intensywność ruchu mierzoną w liczbie pojazdów na sekundę.

Intensywność jest uśredniania przy każdym odczycie (odczyt jest dokonywany po każdej wykonanej akcji). Aby policzyć wartość uśrednioną intensywności należy skorzystać z poprzednio obliczonej intensywności oraz wartości, która została właśnie odczytana z urządzenia pomiarowego. Łatwo wyprowadzić taką zależność:

$$average_N = \frac{\sum_{i=1}^{N} a_i}{N} = \frac{\sum_{i=1}^{N-1} a_i + a_N}{N} = \frac{\sum_{i=1}^{N-1} a_i}{N} + \frac{a_N}{N} = \frac{\sum_{i=1}^{N-1} a_i}{N-1} \times \frac{N-1}{N} + \frac{a_N}{N} = average_{N-1} \times \frac{N-1}{N} + \frac{a_N}{N}$$

Wartość a_N jest nowo odczytaną wartością z urządzenia. Jest ona obliczana w następujący sposób:

$$a_N = \frac{liczba\ pojazdów,\ które\ przyjechały\ podczas\ trwania\ poprzeniej\ akcji}{czas\ trwania\ \ poprzedniej\ akcji}$$

W normalnej sytuacji agent pobiera dane o środowisku z urządzeń pomiarowych, następnie ustala odpowiednią strategię dla danej sytuacji (długości palenia się zielonego światła dla poszczególnych faz). Strategia jest następnie wprowadzana w życie za pomocą kontrolerów sygnalizacji świetlnej.

Zaproponowane rozwiązanie będzie się opierało na metodzie Q-learning. Aby ułatwić dalsze rozważania nad algorytmem uczenia oraz kolejnymi elementami RL wprowadzamy słownik używanych terminów:

- LC (line count) – liczba pasów dojazdowych do danego skrzyżowania
- CVC_i (current vehicle count) liczba pojazdów oczekujących na przejazd na i-tym pasie dojazdowym ($0<i\leq LC$)
- CI_i (current intensity) – aktualna intensywność na i-tym pasie (sposób jej obliczania został podany w poprzednim rozdziale)
- S (states) – zbiór wszystkich możliwych stanów, w których może się znaleźć obiekt
- A (actions) – zbiór wszystkich możliwych akcji, które może wykonać agent
- d(A) (action duration) – czas trwania akcji
- GT_{max} - maksymalny czas palenia się zielonego światła liczony w sekundach

- GT_i (green time) – czas palenia się zielonego światła na i-tym pasie ($0 \leq GT_i \leq GT_{max} \, dla \, i \leq LC$)
- Strategia Eksploracji – strategia wyboru akcji w procesie nauki, z jednej strony powinna próbować jeszcze nie używanych akcji, z drugiej wybierać i uczyć algorytmu tych, które dają dobre rezultaty. Znalezienie złotego środka jest elementem kluczowym przy definiowaniu strategii eksploracji
- Strategia Eksploatacji – strategia używana do wyboru akcji przez algorytm, który jest już po procesie nauki

7. Funkcja wzmocnienia (Reinforcement function)

Funkcja wzmocnienia jest bardzo ważnym elementem RL, ponieważ wyznacza kierunek nauki. Od niej w dużej mierze będzie zależało, czy algorytm dobrze nauczy się wybierać akcje w zależności od stanu środowiska.

Jest wiele możliwych definicji funkcji wzmocnienia. Najważniejsze jest to, żeby jak najlepiej definiowała problem i pozwalała na naukę algorytmowi. Ponieważ w rozważanym złożonym problemie zarządzania ruchem drogowym głównym celem jest oczyszczenie skrzyżowań z pojazdów, każda akcja, dzięki, której zmniejszy się liczba samochodów powinna zostać nagrodzona. Należy też zwrócić uwagę na stan, w którym akcja została podjęta. Akcja, która przynosi dobre wyniki wykonana w stanie pierwszym wcale nie musi być dobra dla stanu drugiego.

Definiujemy funkcję R(s, a), gdzie „s” jest stanem, w którym podjęta jest akcja „a” jako różnicę między liczbą pojazdów w stanie „s”, a stanie „s'”, do którego doprowadziła akcja. Ta różnica jest liczona na podstawie danych z urządzeń pomiarowych, zarówno przed wykonaniem akcji jak i po jej wykonaniu.

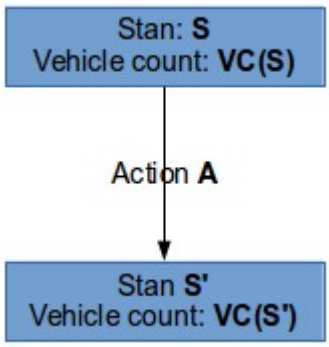


Illustration 24: Wykonanie akcji A w stanie S doprowadza do stanu S'.

Załóżmy, że w stanie S została podjęta akcja A. W stanie S urządzenia pomiarowe odnotowały VC(S) pojazdów czekających na przejazd przez rozważane skrzyżowanie. Wykonanie akcji A doprowadziło do nowego stanu S', w którym na skrzyżowaniu znajduje się VC(S') pojazdów.

Funkcję R(s, a) w rozważanym przypadku definiujemy jako:

$$R(s,a) = VC(S) - VC(S')$$

Przykład:

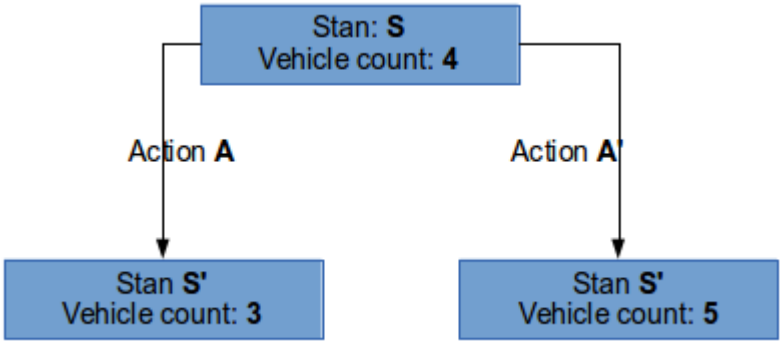
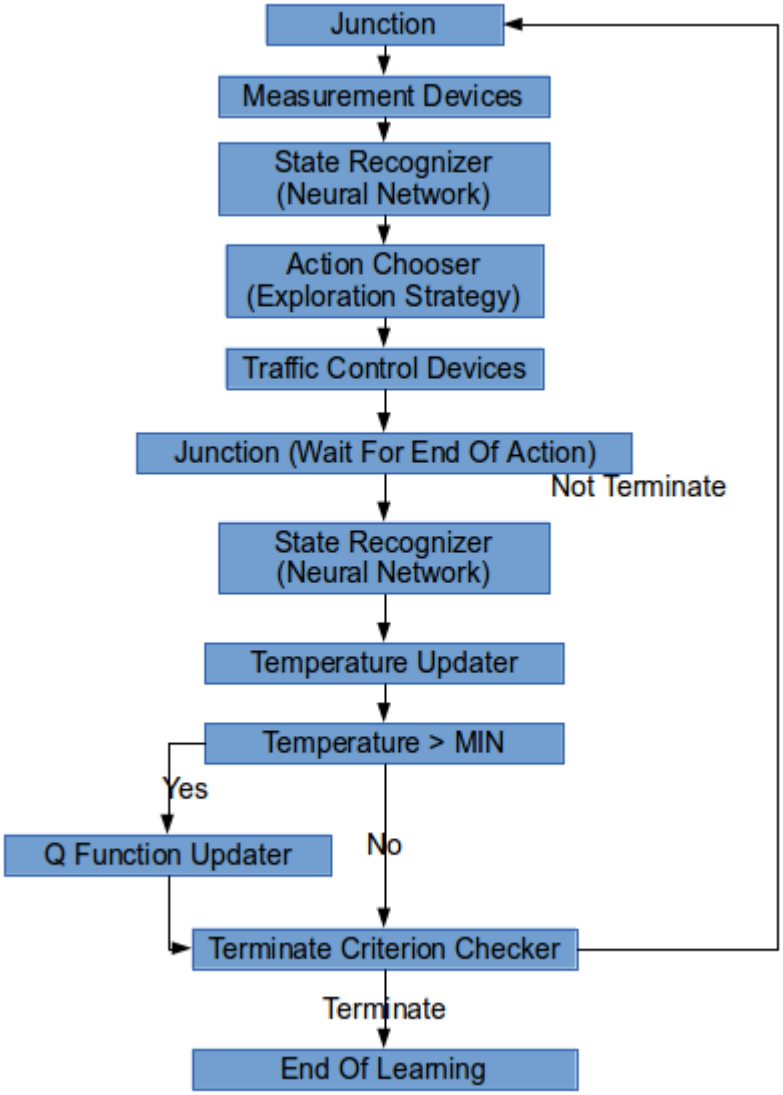


Illustration 25: W stanie S są możliwe akcje A oraz A'. Akcja A jest lepsza, ponieważ doprowadzi do stanu, w którym liczba pojazdów zostanie zmniejszona.

W stanie S liczba pojazdów wynosi 4. Do wyboru są akcje A oraz A'. Po podjęciu akcji A w kolejnym stanie liczba pojazdów będzie równa 3, zatem zgodnie ze wzorem R(S,A) = 4 – 3 = 1. W przypadku akcji A', wartość funkcji będzie wynosiła -1 (R(S, A') = 4 – 5 = -1). Ponieważ 1 > -1, podjęcie akcji A powoduje wzmocnienie (akcja A jest lepszą akcją od A' w stanie S).

8. Proces uczenia

Proces uczenia jest jednym z dwóch kluczowych elementów. W tym podrozdziale opiszemy z jakich elementów się składa i jak przebiega.



Rysunek 4: Schemat proponowanego przez nas procesu uczenia systemu. Proces ten jest podzielony na moduły, z których każdy pełni odmienną rolę podczas uczenia.

Każdy z elementów ma ściśle określoną rolę. W kolejnych podrozdziałach przybliżymy każdy z modułów z bliższej perspektywy.

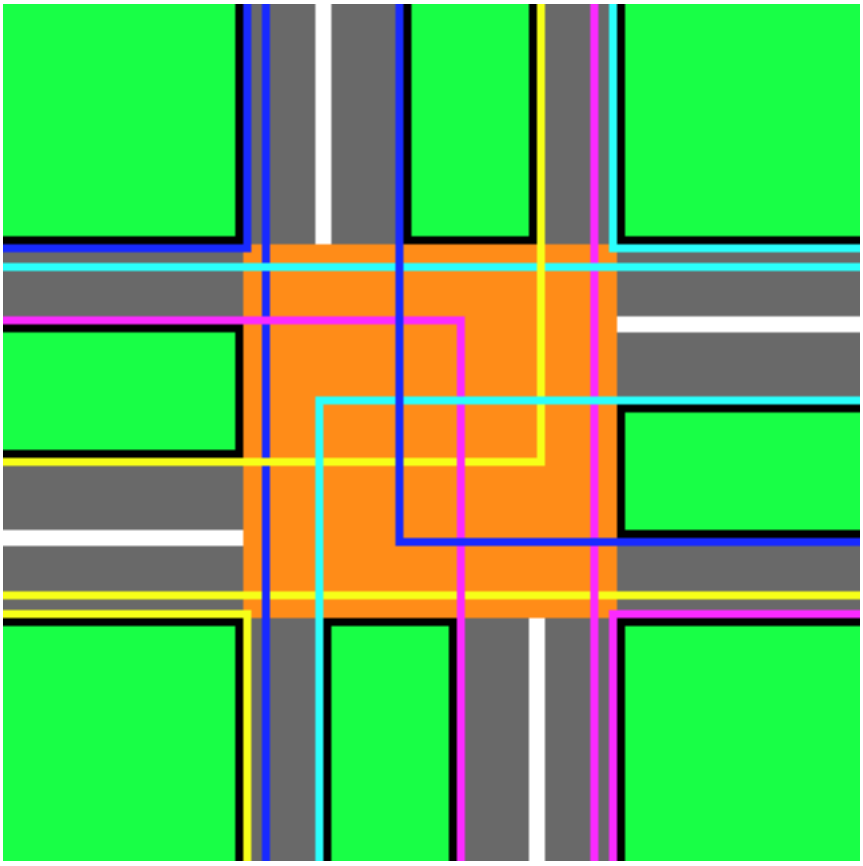
8.1. Skrzyżowanie (Junction)

W naszej pracy będziemy rozważali pojedyncze skrzyżowanie o pasach dojazdowych z zachodu, wschodu, północy oraz południa. Każdy dojazd składa się z dwóch pasów, jeden, do jazdy w lewo, drugi do jazdy na wprost oraz do skrętu w prawo.

Skrzyżowanie będzie pełniło rolę środowiska lub inaczej otoczenia. Agent (algorytm sterujący ruchem) może się dowiadywać o stanie skrzyżowania na podstawie odczytów z sensorów:

- intensywność ruchu na każdym pasie dojazdowym
- liczba pojazdów na każdym z pasów dojazdowych
- suma czasów oczekiwania pojazdów na zielone światło (parametr poglądowy nie wykorzystywany przez agenta)

- suma czasów podróży pojazdów (parametr poglądowy nie wykorzystywany przez agenta)



Rysunek 5: Skrzyżowanie testowe. Kolorami są oznaczone trasy możliwe trasy pojazdów.

Zachowanie pojazdów na skrzyżowaniu jest oparte o zmodyfikowany automat komórkowy VDR.

Automaty komórkowe są strukturami opisanymi przez siatkę komórek, które są opisane za pomocą stanów. Automat komórkowy definiuje w jaki sposób komórki mogą zmieniać swoje stany. Istnieją pewne reguły, które systematyzują zachowanie całego automatu komórkowego.

Najczęściej automat komórkowy składa się z n-wymiarowej siatki, składającej się z takich samych komórek. Są one styczne do siebie. Każda z komórek ma skończony zbiór stanów, w którym może się znaleźć. To, w którym stanie znajdzie się za chwilę jest zdeterminowane przez reguły, o których pisaliśmy wcześniej.

Automaty komórkowe mają wiele zastosowań. Bardzo popularna jest tak zwana „gra w życie", która symuluje ewolucję. Automaty komórkowe zostały także zastosowane w „Mrówce Langtona", „Turmity" oraz „Wireworl".

My chcielibyśmy zastosować automaty komórkowe do symulacji ruchu drogowego. W naszym przypadku bardzo ważne jest modelowanie realistycznego zachowania się samochodów, stąd automaty komórkowe wydają się idealnym rozwiązaniem.

Bardzo dobry automat komórkowy zaproponował Nagel-Schreckenberg. W tym podrozdziale pokrótce opiszemy ideę tego automatu. Będzie on zastosowany w naszej pracy z pewnymi modyfikacjami. Model Nagela, zwany w skrócie NaSch wydaje się spełniać wymogi realizmu, brakuje mu jednak pewnych czynników, które chcielibyśmy badać w naszym symulatorze, stąd konieczność pewnych rozszerzeń i usprawnień.

Opis algorytmu oparty jest na publikacji naukowej „Cellular automata for traffic simulation

Nagel-Schreckenberg model" autorstwa Torstena Held'a oraz Stefana Bittihn'a opublikowanej 17 marca 2011 w Bonn [8].

Model NaSch jest modelem probabilistycznym. Zawiera zatem elementy, losowe. Losowość jest niezbędna przy symulacji rzeczywistego świata. Może być jednak przeszkodą w przypadku porównywania algorytmów, ponieważ powinny zostać one testowane przy jednakowych warunkach i natężeniu ruchu. Stąd chcemy przetestować nasz algorytm w warunkach deterministycznych, gdzie wszystkie elementy ruchu są ściśle zdefiniowane. Na tej podstawie będziemy mogli zobaczyć rzeczywiste różnice między opracowanym przez nas algorytmem a algorytmem standardowym. Dodatkowo dla sprawdzenia efektywności algorytmu, będziemy go testować w środowisku niedeterministycznym, aby zobaczyć jak będzie reagował na nieoczekiwane wydarzeni, które przecież zdarzają się bardzo często na drogach.

Poruszanie pojazdów jest podzielone na 4 kroki:

1. Przyspieszenie (acceleration): $v_n \rightarrow min(v_n+1, v_{max})$
2. Zwolnienie (deceleration): $v_n \rightarrow min(v_n, d_n-1)$
3. Losowość (randomization): $v_n \rightarrow max(v_n-1, 0)$
4. Ruch (movement): $x_n \rightarrow x_n+v_n$

Gdzie:

- v_n - aktualna prędkość
- d_n - odległość od pojazdu z przodu
- x_n - aktualna pozycja

Model może mieć zamknięte lub otwarte granice. Otwarte oznaczają, że pojazd z prawdopodobieństwem α może pojawić się na mapie (pod warunkiem, że pierwsza komórka jest pusta) oraz z prawdopodobieństwem β może opuścić mapę, jeżeli zbliży się do granicy automatu skończonego.

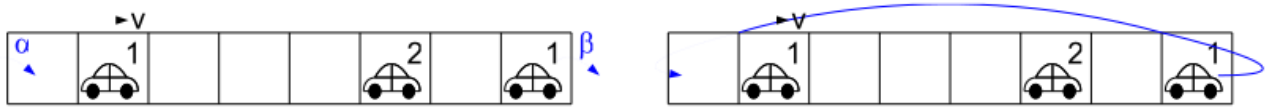


Illustration 26: Po lewej stronie przedstawiona jest idea otwartych granic. Prawa strona przedstawia model NaSch z zamkniętymi granicami.

Zamknięte granice tworzą pętlę. Pojazd, który zbliża się do granicy automatu, zostaje przeniesiony na jego początek.

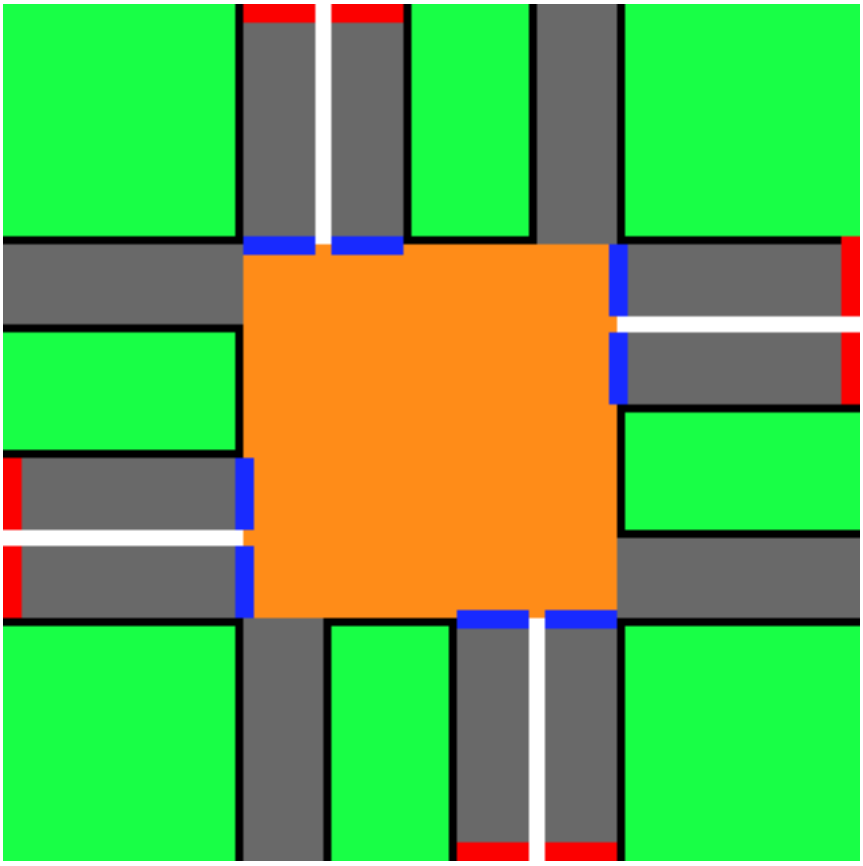
Rozszerzeniem modelu NaSch jest model VDR (velocity dependent randomization). Model jest rozszerzony o zmiany ruchu w zależności od jego gęstości. Dla przykładu, pojazdy poruszają się wolniej, gdy gęstość jest duża. Dodatkowym czynnikiem jest także fakt, że pojazd, który całkowicie się zatrzymał będzie się rozpędzał wolniej.

Według autora model jest dobrym odwzorowaniem obserwowanego na co dzień ruchu drogowego. Rozszerzony o nowe parametry i funkcjonalności może być pomocny w symulowaniu różnego rodzaju zależności i czynników na drodze, takich jak brak koncentracji kierowców, czy opóźnione ruszanie na światłach.

8.2. Urządzenia pomiarowe (Measurement Devices)

Urządzenia pomiarowe są nieodłącznym elementem systemu. Detektory zliczają liczbę pojazdów

wjeżdżający na dany pas ruchu. W rzeczywistym systemie muszą być zamontowane w odpowiednich miejscach na skrzyżowaniu. Dzięki nim można pobierać informacje o skrzyżowaniu takie jak intensywność ruchu czy liczba pojazdów oczekujących na przejazd. Istnieją dwa rodzaje detektorów: wejściowe (entry detectors) oraz wyjściowe (exit detectors). Detektor wejściowy zwiększa licznik pojazdów na danym pasie ruchu oraz wylicza inne potrzebne wartości takie jak intensywność ruchu. Z kolei detektory wyjściowe są umieszczone w taki sposób, aby dekrementować licznik, gdy pojazd opuści już skrzyżowanie.



Ilustracja 27: Czerwonym kolorem zaznaczone są detektory wejściowe, natomiast kolor niebieski symbolizuje detektory wyjściowe.

Detektory są sparowane. Każdemu urządzeniu wejściowemu przypisane jest dokładnie jedno urządzenie wyjściowe.

Detektor wejściowy dodatkowo oblicza intensywność ruchu zgodnie ze wzorem podanym wcześniej.

Bez urządzeń pomiarowych nie byłoby możliwe zbudowanie systemu sterującego ruchem. Są czymś w rodzaju zmysłów systemu, z pomocą których może on się uczyć i poznawać prawa rządzące skrzyżowaniem.

8.3. Klasyfikator stanów (State Recognizer)

W tym podrozdziale opisany zostanie moduł odpowiedzialny za rozpoznanie stanu skrzyżowania. W pierwszej kolejności musimy zdefiniować, czy jest stan skrzyżowania. Można do tego zagadnienia podejść na wiele różnych sposobów.

Stan skrzyżowania jest przedstawiony za pomocą wektora, który posiada $2 \times LC$ elementów, gdzie LC (lines count) jest liczbą pasów dojazdowych do skrzyżowania. Każdy pas jest opisany dwoma parametrami:

1. CVC (current vehicle count) – aktualna liczba pojazdów oczekujących na przejazd na danym pasie
2. CI (current intensity) – aktualna intensywność ruchu na danym pasie

Zbiór wszystkich możliwych stanów musi być zamknięty, stąd potrzeba ograniczenia wartości CVC oraz CI poprzez CVC_{max} oraz CI_{max} . Te ograniczenia powinny być dostosowane do wielkości danego skrzyżowania:

$$CVC_i = \begin{matrix} CVC_i, dla\, CVC_i < CVC_{max} \\ CVC_{max}, w\ pozostałych\ przypadkach \end{matrix}$$

$$CI_i = \begin{matrix} sufit(CI_i), dla\, CI_i < CI_{max} \\ CI_{max}, w\ pozostałych\ przypadkach \end{matrix}$$

Definicja: $sufit(x) = min(k \in \mathbb{Z} : k \geq x)$

Wektor stanu będzie zatem wyglądał następująco:

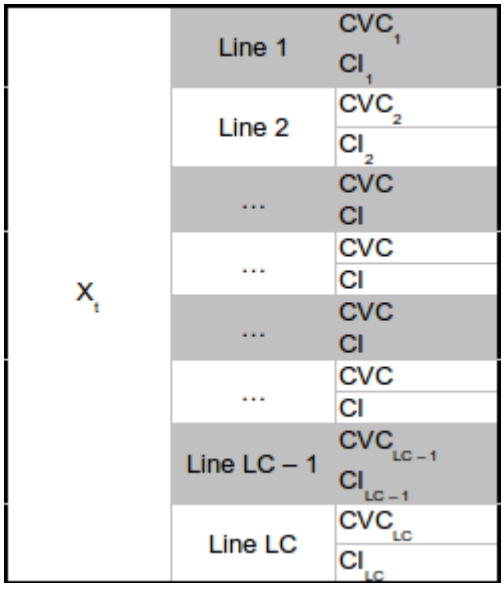


Illustration 28: Wektor stanu dla skrzyżowania w czasie „t" z LC pasami dojazdowymi.

Moc zbioru wszystkich stanów jest bardzo łatwa do policzenia:

$$|S| = ((CVC_{max} + 1) \times (CI_{max} + 1))^{LC}$$

Dodajemy jedynki, ponieważ zarówno CVC jak i CI mogą być zerowe.

W tym miejscu chcemy zaznaczyć, że takie podejście jest bardzo złożone obliczeniowo, stąd dla dużych skrzyżowań oraz wartości granicznych CVC_{max} oraz CI_{max} otrzymujemy bardzo dużą moc zbioru S.

Rozważmy zatem jaki wpływ mają poszczególne czynniki na ogólną liczbę stanów. Załóżmy, że

CVC_{max} i CI_{max} są stałe. Niech S(LC) będzie funkcją jednej zmiennej. Otrzymujemy funkcję rosnącą wykładniczo, którą można sprowadzić do prostszej postaci:

$S(LC) = A^{LC}$, gdzie A jest dowolną liczbą naturalną. Poniższa tabela przedstawia przykładowe wartości tej funkcji dla wybranych parametrów A.

A/LC	2	3	4	5	6	7
2	4	8	16	32	64	128
3	9	27	81	243	729	2187
4	16	64	256	1024	4096	16384
5	25	125	625	3125	15625	78125
6	36	216	1296	7776	46656	279936
7	49	343	2401	16807	117649	823543

Ilustracja 29: Tabela pokazuje jak szybko zwiększa się liczba wszystkich możliwych do osiągnięcia stanów.

Tabela przedstawia niepokojącą własność. Liczba wszystkich możliwych do osiągnięcia stanów rośnie wykładniczo. Dla przykładu dla wartości parametru A równej 7 oraz 7 pasów dojazdowych otrzymujemy 823543 stany. Macierz Q będzie miała zatem aż tyle wierszy. Uczenie takiej macierzy może potrwać bardzo długo, a dodatkowo program będzie musiał wykonywać obliczenia na bardzo dużej macierzy, co wpłynie na jego wydajność.

Uważamy, że tak duża liczba stanów nie jest konieczna do działania algorytmu i definiowania macierzy wartości Q. Stąd pomysł zastosowania sieci neuronowej, która mogłaby określać w jakim stanie znajduje się aktualnie skrzyżowanie.

8.4. Wybór akcji (Action Chooser)

W tym podrozdziale zajmiemy się zdefiniowaniem pojęcia akcji oraz opiszemy działanie i funkcje modułu wybierającego akcje w procesie uczenia.

Akcja jest to pewne działanie, którego celem jest zmiana stanu skrzyżowania. Jest ona zdefiniowana jako LC elementowy wektor liczb naturalnych ograniczonych przez GT_{max} .

a_t	Line 1	GT_1
	Line 2	GT_2
	Line 3	GT_3
	…	GT
	…	GT
	…	GT
	…	GT
	…	GT
	…	GT
	…	GT
	…	GT
	…	GT
	…	GT
	Line LC – 2	GT_{LC-2}
	Line LC – 1	GT_{LC-1}
	Line LC	GT_{LC}

Illustration 30: Wektor akcji w czasie "t" dla skrzyżowania z LC pasami dojazdowymi.

Bardzo łatwo jest policzyć liczbę wszystkich możliwych do uzyskania akcji:

$$|A| = (GT_{max} + 1)^{LC} \quad .$$

Wprowadzamy także tak zwaną długość akcji oznaczaną przez d(A):

$$d(A) = \max_i (GT_i), \; gdzie \; 0 < i \leq LC$$

Należy zwrócić szczególną uwagę na fakt, że niektóre z akcji są niedozwolone. Dzieje się tak dlatego, że dwa pasy mogą być kolizyjne a mimo to będą miały razem zapalone zielone światło. Stąd potrzeba wykluczenia akcji niedozwolonych. W ten sposób zmniejsza się nie tylko ich liczba, ale także rozmiar macierzy Q.

Wybór akcji wiąże się ściśle ze strategią eksploracji. Musimy być świadomi, że w procesie uczenia musimy wykonywać wiele razy te same akcje, aby system mógł odpowiednio zaktualizować wartości funkcji Q. Ważne jest również, aby wybierać różne akcje, a nie wciąż te same. Stąd właśnie pojawia się pojęcie strategii eksploracji, która określa sposób wyboru akcji w procesie uczenia.

Jak podaje autor [7] istnieje wiele rodzajów wyboru akcji. Są to między innymi:

- strategia zachłanna
- strategia ϵ zachłanna
- strategia softmax

Strategia zachłanna charakteryzuje się wyborem akcji, która aktualnie posiada największą wartość w tabeli Q. Takie rozwiązanie ma jednak podstawową wadę. Nie pozwala na przeszukanie całego zbioru akcji w poszukiwaniu najlepszej. W związku z tym, rozszerzono tą strategię o tak współczynnik ϵ. Określa on stopień prawdopodobieństwa zachłannego wyboru akcji. Oznacza to, że z prawdopodobieństwem ϵ zostanie użyta strategia zachłanna, a z prawdopodobieństwem $1-\epsilon$ akcja zostanie wylosowana bez względu na jej wartość w tabeli Q.

Trzecią strategią, którą zastosujemy w naszym rozwiązaniu jest strategia „SOFTMAX”. Ideą tej strategii jest stworzenie rankingu akcji. Dalej, strategia, która posiada największą szacowaną wartość ma największe szanse na wybranie. Im mniejsza wartość w tabeli Q dla akcji, tym mniejsza szansa na jej wybór.

Zgodnie z [7] prawdopodobieństwo wyboru akcji można zdefiniować w następujący sposób:

$$P(a)=\frac{\left(e^{\frac{Q_t(a)}{\gamma}}\right)}{\left(\sum_{b=1}^{N} e^{\frac{Q_t(b)}{\gamma}}\right)} \quad ,$$

gdzie „a” jest daną akcją, N jest liczbą wszystkich akcji, natomiast γ jest dodatnim parametrem określającym różnice między prawdopodobieństwami. Dla wysokich wartości tego parametru, prawdopodobieństwa wyboru zbliżają się do siebie (prawdopodobieństwo wyboru każdej z akcji jest większe). Niska wartość oznacza większe różnice w prawdopodobieństwie wyboru.

Dla przykładu rozważmy trzy akcje z wartościami kolejno 1, 5 oraz 10. W tabeli przedstawiamy prawdopodobieństwa wyboru tych akcji w zależności od parametru γ.

	Akcja 1 (wartość 1)	Akcja 1 (wartość 5)	Akcja 1 (wartość 10)
$\gamma=0.1$	0	0	1
$\gamma=0.5$	0,00000002	0,00004540	0,99995459
$\gamma=1$	0,00012257	0,00669203	0,99318540

Tabela 3: Tabela pokazuje prawdopodobieństwa dla przykładowych akcji oraz różnych wartości parametru gamma.

Przykład pokazuje, że dla większych wartości parametru gamma, prawdopodobieństwa wyboru kolejnych akcji są do siebie zbliżone, podczas gdy niskie wartości parametru oddalają je od siebie,

nobilitując tym samym akcje o większych wartościach.

8.5. Sterownik sygnalizacji świetlnej (Traffic Control Devices)

Sterownik sygnalizacji świetlne koordynuje pracę pojedynczych sygnalizatorów. Dzięki niemu zapalają się zielone lub czerwone światła na pojedynczych urządzeniach. Informacje o cyklach, fazach oraz wszelkich czasach ich trwania są zbierane właśnie w tym module. Na wejście sterownika przychodzi wybrana w poprzednim module akcja. Sterownik na jej podstawie ustawia odpowiedni cykl.

Należy jednak w tym miejscu zwrócić uwagę na pewną różnicę w stosunku do standardowego systemu. Zwyczajowo cały cykl składa się z różnych faz. W naszym rozwiązaniu cykl jest utożsamiany z fazą. Jest to tak zwany cykl jednofazowy. Po jego skończeniu wybierany jest kolejny w oparciu o zdobytą wiedzę.

8.6. Menadżer temperatur (Temperature Updater)

Ten moduł wprowadza pojęcie Temperatury, która określa stopień prawdopodobieństwa przejścia do stanu $S_{(i+1)}$ ze stanu S_i po wykonaniu akcji „a”. Temperaturę wprowadza się w celu wyeliminowania z procesu nauki sytuacji, które są wyjątkami (zdarzają się niezwykle rzadko lub przez przypadek stąd nie powinny być brane pod uwagę w procesie uczenia).

Temperaturę definiuje się w jako:

$$T(S_i, a, S_{i+1}) = \frac{(|S_i, a, S_{i+1}|)}{(|S_i, a|)}$$

Licznik oznacza liczbę akcji „a”, które zostały wywołane w stanie S_i i doprowadziły do stanu S_{i+1}, natomiast mianownik określa liczbę wszystkich wywołań akcji „a” w stanie S_i. Gdy temperatura jest niska ($T < T_{min}$) podjęte działanie zostaje uznane jako mało wiarygodne i nie następuje proces uczenia (aktualizacji funkcji Q). Zaczyna się wtedy kolejna iteracja.

Omawiany w tym podrozdziale parametr jest pewnego rodzaju filtrem, który odróżnia zachowania przypadkowe od standardowych. Sytuacje bardzo rzadkie nie powinny brać udziału w procesie uczenia, aby nie zniekształcać wyników. System powinien raczej szukać punktów stałych i uczyć się standardowych zachowań aniżeli wprowadzać niepotrzebne zaburzenia spowodowane losowymi i rzadkimi incydentami.

8.7. Moduł aktualizacji macierzy Q (Q function updater)

Moduł aktualizacji jak nazwa wskazuje odpowiedzialny jest za aktualizację funkcji Q. Dzięki temu komponentowi funkcja Q będzie dążyła do optymalnej funkcji Q*, która stanowi efekt optymalnego uczenia algorytmu. W rzeczywistości bardzo rzadko osiąga się idealne przybliżenie funkcji Q, zwykle zadowala nas już przybliżenie charakteryzujące się pewnym błędem.

Gdyby funkcja Q* była zawsze osiągalna, strategia zachłanna wyboru akcji byłaby jedną z najlepszych strategii wyboru akcji w nauczonym algorytmie. Niestety, czasem sieć nie jest wystarczająco nauczona. Stąd istnieją sytuacje, gdy wybór akcji niezgodnej ze strategią zachłanną może okazać się lepszy.

Wymaganiem stawianym temu modułowi jest zatem taka aktualizacja funkcji wartości Q, aby ta dążyła do Q*, czyli swojej optymalnej postaci.

Gdy temperatura T osiąga pewną, zadowalającą wartość funkcja Q jest aktualizowana zgodnie ze wzorem zaproponowanym przez Bellmana:

$$Q(x_t, a_t) = r(x_t, a_t) + \gamma \max_a (Q(x_{t+1}, a))$$

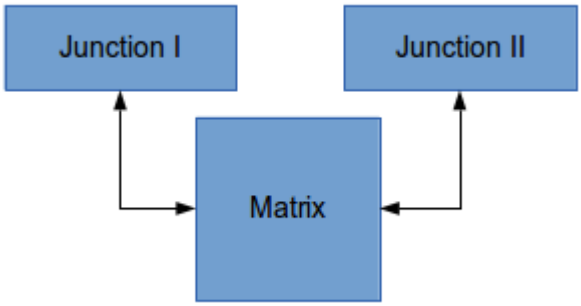
Moduł aktualizacji macierzy Q posiada 2 tryby:

1. Tryb niezależnych macierzy (każde skrzyżowanie posiada osobną macierz, którą tylko ono wykorzystuje)
2. Tryb współpracy (tworzymy dwie niezależne kopie tego samego skrzyżowania i uczymy je równolegle)
    1. Skrzyżowania posiadają jedną macierz Q, którą razem aktualizują
    2. Skrzyżowania wymieniają się sensacjami, to jest, gdy na skrzyżowaniu pierwszym za wykonanie akcji zostanie nadana wystarczająco duża nagroda. Macierze obu skrzyżowań zostaną zaktualizowane o nową wartość.
    3. Co pewien czas następuje uśrednianie obu macierzy

Tryb pierwszy jest najprostszą formą uczenia. Procesie nauczania poddawane jest pojedyncze skrzyżowanie posiadające jedną macierz Q.

W celu przyspieszenia uczenia i próbie eliminacji problemu niedouczenia wprowadziliśmy także drugi tryb oparty na uczeniu dwóch kopii tego samego skrzyżowania. Można z niego wydzielić 3 opcje.

Pierwsza z nich polega na utrzymywaniu tylko jednej macierzy wartości dla obu kopii. Taki zabieg ma na celu poszerzenie zakresu uczenia oraz próbie przyspieszenia tego procesu.



Ilustracja 31: Dwie kopie tego samego skrzyżowania korzystają z jednej tabeli wartości Q.

Inną formą współdziałania jest tak zwane dzielenie sensacji. Załóżmy, że skrzyżowanie pierwsze znajduje się w stanie S_0 , w którym wykonana zostaje akcja A_0 , która doprowadza do stanu S_1 . W wyniku takiego działania, algorytm otrzymuje wzmocnienie większe od R_{min} , które można policzyć ze wzoru:

$R_{min}=\sigma \times R_{max}$, gdzie sigma jest liczbą z przedziału od 0 do 1, natomiast R_{max} jest największą osiągniętą do tej pory wartością wzmocnienia. Gdy otrzymane wzmocnienie będzie zatem wystarczające (spełniające wcześniejszy warunek), pole macierzy Q odpowiedzialne za przechowywanie wiedzy o wykonywaniu akcji A_0 w stanie S_0 zostanie zaktualizowane w macierzy skrzyżowanie pierwszego i drugiego.

Trzecią opcją jest uśrednianie obu macierzy co pewien określony czas. Po tej operacji macierze obu agentów (skrzyżowań) będą takie same. Oznaczmy macierz skrzyżowania pierwszego jako Q_1 , natomiast macierz skrzyżowania drugiego jako Q_2 . Wtedy, po operacji uśredniania powstaje macierz Q_3 , która następnie zastępuje macierz Q_1 oraz Q_2 . Uśrednianie (averaging) działa w następujący sposób:

$Q_3(i,j)=\frac{Q_1(i,j)+Q_2(i,j)}{2}$, gdzie „i” jest numerem wiersza, natomiast „j” jest numerem komórki.

8.8. Moduł kryterium zakończenia (Terminate Criterion Checker)

Jednym z ważniejszych elementów jest ustalenie warunków zakończenia procesu uczenia systemu. Należy zadać sobie pytanie, kiedy system jest już nauczony, a kiedy nauka powinna być kontynuowana.

Uczenie w Q-Learning polega na jak najlepszym przybliżeniu funkcji Q, zatem kolejne wartości powinny być coraz bliższe optymalnym. W naszym rozwiązaniu wartości funkcji Q są przedstawione w postaci tabeli.

		Actions								
		Action 1	Action 2	Action 3	Action 4	Action 5	Action 6	Action 7	Action 8	Action 9
States	State 1	0	0	0	0	0	0	0	0	0
	State 2	0	0	0	0	0	0	0	0	0
	State 3	0	0	0	0	0	0	0	0	0
	State 4	0	0	0	0	0	0	0	0	0
	State 5	0	0	0	0	0	0	0	0	0
	State 6	0	0	0	0	0	0	0	0	0
	State 7	0	0	0	0	0	0	0	0	0
	State 8	0	0	0	0	0	0	0	0	0
	State 9	0	0	0	0	0	0	0	0	0
	State 10	0	0	0	0	0	0	0	0	0
	State 11	0	0	0	0	0	0	0	0	0
	State 12	0	0	0	0	0	0	0	0	0
	State 13	0	0	0	0	0	0	0	0	0
	State 14	0	0	0	0	0	0	0	0	0
	State 15	0	0	0	0	0	0	0	0	0
	State 16	0	0	0	0	0	0	0	0	0
	State 17	0	0	0	0	0	0	0	0	0
	State 18	0	0	0	0	0	0	0	0	0
	State 19	0	0	0	0	0	0	0	0	0
	State 20	0	0	0	0	0	0	0	0	0
	State 21	0	0	0	0	0	0	0	0	0
	State 22	0	0	0	0	0	0	0	0	0
	State 23	0	0	0	0	0	0	0	0	0
	State 24	0	0	0	0	0	0	0	0	0
	State 25	0	0	0	0	0	0	0	0	0
	State 26	0	0	0	0	0	0	0	0	0
	State 27	0	0	0	0	0	0	0	0	0
	State 28	0	0	0	0	0	0	0	0	0
	State 29	0	0	0	0	0	0	0	0	0
	State 30	0	0	0	0	0	0	0	0	0
	State 31	0	0	0	0	0	0	0	0	0
	State 32	0	0	0	0	0	0	0	0	0

Ilustracja 32: Przykładowa tablica wartości funkcji Q. Argumentami tej funkcji jest stan i akcja. Początkowo tabela jest inicjalizowana samymi zerami.

Funkcja Q zostanie szczegółowo opisane w jednym z kolejnych rozdziałów.

Każda komórka tabeli Q (będziemy czasami używali słowa tabela zamiast funkcja) określa wartość akcji wykonanej w pewnym stanie. Ponieważ funkcja Q powinna być zbieżna, każdy z jej elementów będzie zbiegał do pewnej liczby Q_{IJ} , gdzie I jest pewnym stanem, natomiast J akcją. Osiągnięcie granicy funkcji (idealnego przybliżenia) nie jest możliwe, stąd określamy przedział błędu, który nas zadowala. Niestety nie znamy granicy funkcji Q, stąd obliczenie błędu nie może

polegać na różnicy między wartością graniczną a aktualną. Posłużymy się natomiast odchyleniem standardowym definiowanym następująco dla dyskretnego łańcucha wartości:

$$\sigma=\sqrt{(\frac{(\sum_{i=1}^{N}(x_i-\mu)^2)}{(N)})}\ ,$$

gdzie:

- σ - odchylenie standardowe
- μ - wartość oczekiwana (w naszym przypadku średnia arytmetyczna)
- x_i - kolejne wartości cechy
- N – liczba badanych elementów

Jak już wcześniej pisaliśmy funkcja Q może przyjmuje wartości z tabeli Q_{IJ} . Wartości te będą się zmieniały w każdej iteracji (po każdej epoce). Definiujemy wartość cechy x_i jako średnią arytmetyczną z wszystkich wartości Q_{IJ} , gdzie indeks „i” oznacza numer iteracji.

$$x_i=\frac{\sum_{k=0}^{I-1}\sum_{t=0}^{J-1}Q_{IJ}}{(I\times J)}$$

Wartość oczekiwana μ liczona jest ze wzoru:

$$\mu=\sum_{i=1}^{N}x_i$$

Ponieważ nie chcemy przechowywać informacji o wszystkich wartościach x_i ze względu na zbyt duże zużycie pamięci, posłużymy się zależnością wyprowadzoną już wcześniej.

$$\mu_N=\frac{\sum_{i=1}^{N}x_i}{N}=\frac{\sum_{i=1}^{N-1}x_i+x_N}{N}=\frac{\sum_{i=1}^{N-1}x_i}{N}+\frac{x_N}{N}=\frac{\sum_{i=1}^{N-1}x_i}{N-1}\times\frac{N-1}{N}+\frac{x_N}{N}=\mu_{N-1}\times\frac{N-1}{N}+\frac{x_N}{N}$$

Pierwszym warunkiem zakończenia jest osiągnięcie odchylenia standardowego w określonym przedziale. Ustalamy maksymalne odchylenie na poziomie σ_{max} i zdefiniujmy warunek zakończenia jako $\sigma<\sigma_{max}$.

Niestety powyższy warunek może czasami nie zostać nigdy spełniony. Wtedy określamy drugi warunek zakończenia bazujący na liczbie iteracji M. Zwyczajowo ten parametr powinien być bardzo duży, aby algorytm dobrze nauczył się swojej roli.

Zwracamy jednak uwagę, że system, który przestał się uczyć ze względu na spełnienie warunku drugiego (osiągnięty limit iteracji) może wykazywać się słabymi wynikami ze względu na funkcję Q, której wartości znacznie odbiegają od granicznych.

9. Proces eksploatacji

Działanie algorytmu w świecie rzeczywistym jest oparte na wyuczonych wartościach z macierzy Q. Załóżmy, że skrzyżowanie znajduje się w stanie S. Z macierzy Q można odczytać, która akcja będzie najlepsza dla danego stanu.

Należy rozważyć jednak głębiej zagadnienie eksploatacji. Możemy wybrać różne strategie wyboru akcji:

strategia zachłanna – wybierana jest akcja, która ma największą wartość w macierzy Q

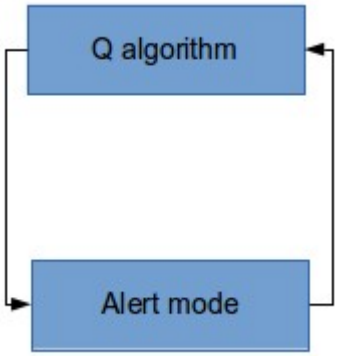
strategia Boltzmanna – akcja z największą wartością nie zawsze musi zostać wybrana. Motywacją do wyboru tej strategi jest możliwość błędnego uczenia sieci, stąd dajemy szansę na wybór mniej wartościowanej akcji, która może w rzeczywistości okazać się lepsza.

Do naszego rozwiązania wybraliśmy strategię Boltzmanna. Prawdopodobieństwo wyboru akcji niezachłannej (czyli takiej, która nie ma największej wartości dla danego stanu) jest tym mniejsze, im bardziej akcja zachłanna jest oddalona (chodzi tutaj o wartości Q) od pozostałych. Prawdopodobieństwo liczone jest ze wzoru:

$$\pi(x,a') = \frac{\exp(\frac{Q(x,a')}{T})}{\sum_{a} \exp(\frac{Q(s,a)}{T})}$$, gdzie T jest temperaturą (T>0), która reguluje stopień losowości.

Im mniejsze T, tym większe prawdopodobieństwo, że zostanie wybrana akcja zgodnie z strategią zachłanną.

Ideą systemu, jest wyprowadzenie skrzyżowania ze stanu kryzysowego (duża liczba oczekujących na skrzyżowaniu samochodów lub długie czasy oczekiwania na przejazd) do stanu normalnego oraz uruchomienie trybu alarmowego, gdy na skrzyżowaniu pojawi się pojazd uprzywilejowany.



Ilustracja 33: Dwa tryby działania: alarmowy i inteligentny.

Tryb alarmowy działa bardzo prosto. Gdy pojazd uprzywilejowany zostanie wykryty na danym pasie zostanie zapalone zielone światło do czasu opuszczenia skrzyżowania przez pojazd. W przypadku pojawienia się dwóch pojazdów uprzywilejowanych zostaną one obsłużone zgodnie z czasem pojawienia się na skrzyżowaniu lub równocześnie, jeżeli sytuacja na o pozwoli (będzie możliwy bezkolizyjny przejazd przez skrzyżowanie obu pojazdów).

VII. Implementacja symulatora
VIII. Rezultaty
IX. Wnioski
X. Zakończenie
XI. Literatura

(1) Ming Tan: Multi-Agent Reinforcement Learning: Independent versus Cooperative Agents,

opublikowane w „Readings in agents”, strony: 487-494, wydawca: Morgan Kaufmann Publishers Inc. San Francisco, CA, USA 1998, ISBN: 1-55860-495-2

(2) Zbigniew Michalewicz: „Genetic Algorithms + Data Structures = Evolution Programms”, ISBN: 3-540-60676-9 Springer-Verlag Berlin Heidelberg New York

(3) Leena Singh, Sundhanshu Tripathi, Himakshi Arora: Time Optimization for Traffic Signal Control Using Genetic Algorithm, Internationl Journal of Recent Trends In Engineering, Vol 2, No. 2.

(4) www.ai.c-labtech.net

(5) Sergey V. Anfilets, Vasilij N. Shuts: Artificial neural networks for adaptive management traffic light object at the intersection.

(6) Reinforcement Learning: A Tutorial by Stephanie S. Harmon

(7) Richard S. Sutton and Andrew G. Barto: Reinforcement Learning – And Introduction

(8) Torsten Held, Stefan Bittihn: Cellular automata for traffic simulation – Nagel-Schreckenberg model, Bonn, 17 marca 2011