Optimizing traffic flow using learning and the shout-ahead agent architecture

Christian Roatis
Department of Computer Science
University of Calgary
Calgary, Canada
christian.roatis@ucalgary.ca

***Abstract*—We present an attempt to improve traffic throughput in the traffic simulator SUMO by implementing the shout-ahead agent architecture and a hybrid behavior learning method for it into traffic signal controller agents. Problems with current traffic signal controllers, namely their inefficient, rigid phase changing algorithms, served as motivation for the project. The performance of the shout-ahead enabled traffic signal control agents is determined over a set of experimental evaluations, using simulation time as a performance measure, and we will compare the shout-ahead traffic signal controller's performance relative to that of a standard signal control algorithm utilized by SUMO to determine the effectiveness of our implementation.**

***Index Terms*—traffic throughput, cooperative systems, optimization, shout ahead**

I. Introduction/Motivation

For modern suburb dwellers in large metropolitan cities across the globe, the drive to and from work, often featuring long commute times, has become a part of life [1]. A 2016 Statistics Canada report found that 7% of all car commuters in Canada spend at least 60 minutes travelling to work, with that number jumping to 10% for car commuters in metropolitan areas [2]. Long commute times are generally a result of traffic jams caused by bottlenecks, rather than long distances, and traffic signal timing at intersections is a major factor in creating said bottlenecks [3]. Traffic signal timing can also hinder traffic flow at times without congestion, such as at night, where for instance vehicles may be held up at an intersection that is giving way to a direction with no oncoming vehicles.

Problems such as these string from traffic signal algorithms that do not effectively take into account traffic volume and flow, and lack the flexibility to adapt to the ever-changing traffic state. Specifically, traffic signal algorithms rely on pre-set timer cycles that can be adjusted based on demand, but do so in predetermined ways that do not optimally accommodate for the current state of an intersection [4]. These pre-set timing cycles are also why vehicles may be held up at an empty intersection in the middle of the night. Adjusting traffic signal behavior based on current traffic states could theoretically improve traffic flow, by jettisoning through heavier flows for longer during rush hour, and responding to demand immediately at empty intersections, to name just a few examples. Theoretically, traffic flows could be optimized further if traffic signal algorithms were able to account for expected traffic flows in the immediate future and adjust their behaviors accordingly. In this paper, we propose a solution embodying such characteristics by implementing the shout-ahead architecture and hybrid behavior learning method for it into traffic signal controller agents.

II. Basic Concepts

In this section, we provide a formal description of the shout-ahead architecture, the hybrid behavior learning method for it, and an introduction to the SUMO traffic simulator, which serves as the application area for this project. Before describing the shout-ahead agent architecture and learning method for it, we must first define what constitutes an agent. An agent $\mathcal{A}g$ is represented as a quadruple $\mathcal{A}g$ = (Sit,Act,Dat,f_{Ag}), where Sit is the set of situations $\mathcal{A}g$ can distinguish (its perceptions), Act is the set of actions $\mathcal{A}g$ can perform, Dat is the set of possible value combinations of $\mathcal{A}g$'s internal data areas (all internal "states" $\mathcal{A}g$ can have) and $f_{Ag} : Sit \times Dat \rightarrow Act$ is $\mathcal{A}g$'s decision function, taking in the current situation and value combination of the internal data areas, and deciding on the next action to take based on them.

A. *Shout-ahead architecture*

For the following, we assume we have a group A of agents in an environment $\mathcal{E}nv$, such that $A = \{\mathcal{A}g, \mathcal{A}g_1, ..., \mathcal{A}g_n\}$ with $\mathcal{A}g = (Sit, Act, Dat, f_{Ag})$, $\mathcal{A}g_i = (Sit_i, Act_i, Dat_i, f_{Ag_i})$ for all i, $1 \leq i \leq n$. At the core of the shout-ahead architecture is the concept of a *rule with weight*, which has the general form

$$\text{IF } cond \text{ THEN } a, w.$$

Naturally, $a \in Act$ (Act will always contain a do-nothing action). The condition $cond$ has to allow for elements from Dat and Sit, accomplished via the set Obs of observations about a situation and the current value of the data areas of Dat that do not hold rules. An element of Obs is a predicate about the environment, other agents, or the agent itself. A condition of a rule is simply a set of elements of Obs, i.e. $cond \subseteq Obs$, and a condition $cond = \{obs_i, ..., obs_m\}$ is true, if each obs_j is true in the current situation $s \in Sit$ and the current value $d \in Dat$. Formally, evaluating the truth-value of rule conditions is performed by the function $f_{eval} : 2^{Obs} \times Sit \times Dat \rightarrow Boolean$. The subset Obs_{int} of

Obs contains all observations that are communicated from other agents in A. These observations indicate what actions these agents intend to, or have already made in s. Thus, Obs_{int} contains all the actions from all the Act_i, but for a given situation s and internal data area value combination d, we have that $f_{eval}(a'_i, s, d) = true$ for at most one $a'_i \in Act_i$, namely the action that $\mathcal{A}g_i$ intends to take in s.

The shout-ahead agent architecture uses two rule sets, the set RS and the set RS_{int}, whose current instantiations are stored in the current value of Dat (in their own data areas). Rules in RS have only elements from $Obs \backslash Obs_{int}$ in their conditions, while rules from RS_{int} have only elements from some subset $Obs_{coop} \subseteq Obs$ with $Obs_{int} \subseteq Obs_{coop}$. Therefore, rules in RS have no observations about intended actions and rules in RS_{int} do. Additionally, rules in RS_{int} can contain other observations from Obs in addition to communicated intentions. The decision function f_{Ag} then makes use of these two rule sets to determine what action the agent will take. It first computes $\mathcal{A}g$'s intended action and communicates it to other agents, then uses the communications from those agents to make a final decision on what Ag's final action will be. Therefore, reflecting the two rule sets, f_{Ag} essentially consists of two sub-functions, f_{Ag}^{int} and f_{Ag}^{fin}. f_{Ag}^{int} separates all rules in RS that have conditions evaluating to true in s into two sets, one holding all valid rules with maximal weight, the other holding the rest, and then chooses one rule from these two sets probabilistically; these probabilities are formally defined in [5]. f_{Ag}^{fin} works the same way, except it operates on RS_{int} instead. Then, $\mathcal{A}g$ takes the action of one of these two rules, the selection process involving a mixture of using probabilities and considering the weights of the rules. The probability parameter p_{coop} determines how much the agent relies on the shout-ahead intentions of other agents. If in any situation there are no rules in any of the rule sets in Dat with conditions evaluating to true, the agent takes the action that does nothing.

B. Hybrid behaviour learning method for shout-ahead

The hybrid learning method for cooperative behaviour for the shout-ahead architecture contains two nested loops: one inner loop where simulation runs facilitate reinforcement learning, and an outer loop, where the evolutionary part of the hybrid learning method operates. An agent $\mathcal{A}g$ belongs to a pool, which itself represents the population of agent strategies for $\mathcal{A}g$. There exists an agent pool for every type of role present in the scenario of a simulation. For evaluating individuals within a given pool, each is applied in several simulations and during each simulation run, agents apply reinforcement learning of the weights of their rules. Formally, we assume we have a set $APS = (AP, AP_1, ..., AP_k)$ of agent pools such that AP contains possible strategies (individuals) for $\mathcal{A}g$ and for each $\mathcal{A}g_i \in A$, there is an agent pool AP_j such that AP_j contains possible strategies for $\mathcal{A}g_i$. For a scenario $Scen$, we have as *team requirement* TR a certain number of agents from some agent pools, i.e. $TR = (AP'_1, ..., AP'_l)$, with $AP'_j \in \{AP, AP_1, ..., AP_k\}$. A simulation team $st = (ag_1, ..., ag_l)$ needs to fulfill the team requirements TR, i.e. $ag_j \in$AP$'_j$. A simulation team is used in a simulation run sr of the scenario $Scen$. The result of sr is on the one hand side a run fitness result $rfit(sr, ag_j)$ for each of the agents of st, and also an update of the weights of the rules for each ag_j that is based on the agent architecture. Rule weights are updated performing a variant of the Sarsa reinforcement learning method (see [6]). For each action performed in the simulation run, we update the weight of the rule responsible for it in the following way. If rl is the rule and w its weight and if the reward is r and w' is the weight of the rule responsible for the action following the action from rl, then w is updated by

$$w \quad \leftarrow w + \alpha[r + \gamma w' - w]$$

where γ, $0 \leq \gamma \leq 1$, is the discount rate and α, $0 \leq \alpha \leq 1$, is the learning factor. Then, to create the next generation of strategies for $\mathcal{A}g$, the usual genetic operators crossover and mutation are applied to the strategies, though with a twist, since available rule weights from reinforcement learning are considered in the process. Thus, given two strategies (RS, RS_{int}) and (RS', RS'_{int}), a crossover creates the strategy $(RS^{new}, RS_{int}^{new})$ by selecting the $|RS|$ best rules (with regard to their weights) from $RS \cup RS'$ to create RS^{new} and the $|RS_{int}|$ best rules from $RS_{int} \cup RS'_{int}$ to create RS_{int}^{new}. If RS^{new} or RS_{int}^{new} contain copies of the same rule with different weights, the rule with the lower weight is mutated. Mutation involves either taking in a rule, deleting a random set of observations from $cond$ and adding another randomly chosen set of observations, or creating a new strategy by randomly choosing a rule in either RS or RS_{int} and mutating it as described previously. Finally, the fitness fit of a strategy (RS, RS_{int}) is created out of applying the strategy in several simulation runs $sr_1, ...sr_{s_{max}}$ with different simulation teams, combining the $rfit(sr_i, ag)$-values. It follows that some of the recieved rewards by rules will be part of the value computed by $rfit$, as will be the overall result and time of a simulation run, but the concrete implementation of $rfit$ and fit depends on the application area.

C. SUMO - Simulation of Urban MObility

Short for "Simulation of Urban MObility", SUMO is an open source, microscopic traffic simulator, developed for the purpose of aiding vehicular transportation related research [7]. Highly extendable and modification friendly, SUMO allows users to create and manipulate road networks, traffic flow, traffic signal algorithms and much more, making it an ideal platform for this project. Additionally, SUMO offers a plugin called TraCI (Traffic Control Interface), which uses a TCP based client/server architecture to provide runtime access to road traffic simulations, allowing for the value retrieval and behaviour manipulation of simulated objects on line. TraCI, and by extension SUMO, allows interfacing from Python, meaning complex systems, like a traffic signal controller agent architecture, can be built to run in the background with inputs from a SUMO simulation, manipulating simulated objects based on its outputs.

SUMO provides two primary default signal-timing algorithms that are similar to the preset ones used in intersections today [4] [8]. One asserts that all traffic lights work on a fixed cycle, with a default cycle time of 90 seconds. That is, all possible phases in an intersection are completed within the 90 second interval before beginning again. The cycle time can be adjusted by the user. The other signal-timing algorithm built into SUMO involves "Actuated Traffic Lights" (ATL) [8]. This algorithm supports gap-based actuated traffic control, which allows for phase cycle lengths to adapt to dynamic traffic conditions. SUMO's ATL algorithm uses set phase cycles like the first signal timing algorithm we described above, but if a continuous traffic stream is detected at the end of a green phase, for instance, that phase is prolonged until a sufficient time-gap between vehicles is detected. The purpose of this is to more effectively distribute green phases based on demand, affecting cycle durations based on traffic conditions as a result. Such a signal control scheme is common in Germany, and is similar to more modern systems used around the world [4]. Additionally, SUMO allows for the importing of third-party signal-timing algorithms. [8].

III. Proposed Solution to the Problem

In this section, we present our proposed solution to the traffic signal controller problem via the rule-based shout-ahead agent architecture and accompanied hybrid behavior learning method detailed in [5].

A. *Instantiating the shout-ahead architecture*

In the following, every set of individual traffic signals in an intersection is controlled by an agent, which is responsible for changing their states. Unlike the shout-ahead enabled agents in [5], traffic signal controller agents deal with a completely different set of actions in a completely different environment, so adjustments and improvisations had to be made to fit the architecture to this application area. Firstly, within SUMO, we must define what "roles" we will make available to our agents. Within the context of this application area, a role constitutes a type of intersection. For the purpose of this project, we instantiate three types of intersections, and thus three different classes of intersection controller agents: (1) Standard 4-arm Intersection: Fig. 1 for example, is an obvious choice, given its prevalence in road networks globally, (2) T- Intersection: Fig. 2 for example, is another prevalent intersection in road networks. Much like the Standard 4-arm Intersection, it is simple in design but presents many interesting test cases given the forced turning scenario presented by one of its straight phases ending, (3) 4-arm Incoming Layout Intersection: Fig. 3 for example.

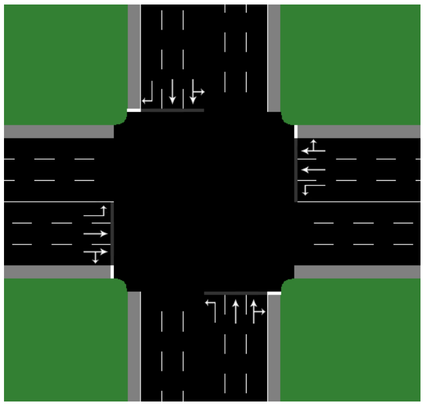
Fig. 1. Example of a standard 4-arm intersection in SUMO.

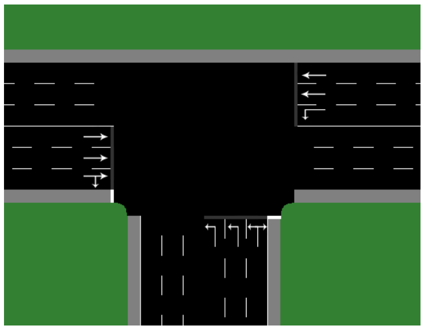
Fig. 2. Example of a T-intersection in SUMO.

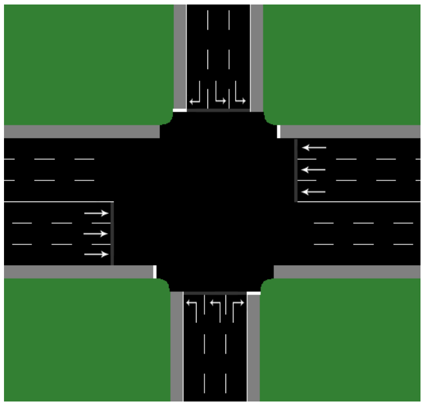
Fig. 3. Example of a 4-arm Incoming Intersection in SUMO.

Unlike the other two, this intersection type is more complex, and potentially presents more bottlenecks, or inefficient signal timing opportunities. The design of this intersection is more irregular than the previous two selected, with a two-way road having two separate one-way roads converging in on it orthogonally, requiring at a minimum three separate green phases. This is unlike the Standard 4-arm and T-Intersections, whom theoretically only require a minimum of two green phases, should they choose to exclude dedicated left-turning phases. In real world applications, this intersection type is far less prevalent than the other two intersections, so its inclusion in our study allows us to see the effectiveness of the shout-ahead architecture when applied to potentially more complex, irregular scenarios. As mentioned previously, within the shout-ahead architecture, an agent holds two rule sets, each containing a number of rules with corresponding actions that the agent can apply at a given time-step. We however extended this concept, adding two additional rule sets, RS^{exp} and RS^{exp}_{int}, which contain user defined exception rules. Rules within this set have maximum priority, such that if a rule within this set is applicable at a given time step, it will always be chosen. If there are multiple applicable rules within this set at a given time, the one with the highest weight will be chosen. If no rules within this set are applicable, then the selection process will continue as described previously. This allows for prioritization of both rare, but critical circumstances such as giving way to an emergency vehicle, and rules we may deem necessary in certain circumstances.

To instantiate the shout-ahead architecture for our problem case, we must first determine how we define our environment Env, namely how we populate the set Obs. As such, we instantiate 81 predicates, to be used in $cond$ for rules in RS. 57 of these are range predicates, meaning they pertain to a certain range of values. This is done because information about Env, an intersection, includes data on waiting times and vehicle counts, which can best be represented in this manner. Thus, these 57 predicates actually serve as sub-predicates to 5 general ones: (1) LongestTimeWaitedToProceedStraight, which ranges from 0 to 300 seconds over 14 sub-predicates, (2) LongestTimeWaitedToTurnLeft, which ranges from 0 to 300 seconds over 14 sub-predicates, (3) NumOfCarsWaiting-ToProceedStraight, which ranges from 0 to 45 vehicles over 8 sub-predicates, (4) NumOfCarsWaitingToTurnLeft, which ranges from 0 to 15 vehicles over 7 sub-predicates and (5) TimeSpentInCurrentPhase, which ranges from 0 to 300 seconds over 14 sub-predicates. The other predicates are concerned with the state of the traffic light signals in the intersection. The state of a traffic light is stored in Dat, and is comprised of three components: (1) the direction traffic is arriving from, including "vertical" and "horizontal" for traffic flowing bilaterally, or "north-south", "south-north", "east-west" or "west-east" for traffic flowing in only one direction, with the destination (2) the phase, which describes whether traffic will flow "straight" (and naturally, "right" as well), such as in Fig. 4, "left", such as in Fig. 5, or "straight left" (indicating both straight and left flow is permitted) during it, such as in Fig. 6, and (3) the state of the traffic signal, which is either "green" or "yellow". A predicate exists for every combination of direction, phase and signal state, so to account for any type of intersection.

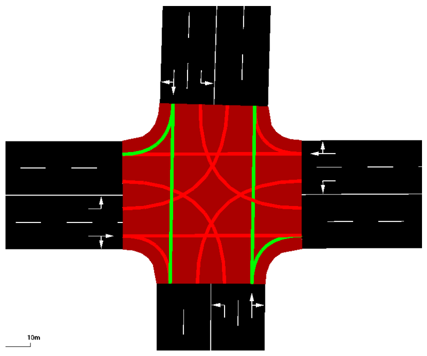

Fig. 4. Example of a "straight phase" in SUMO.

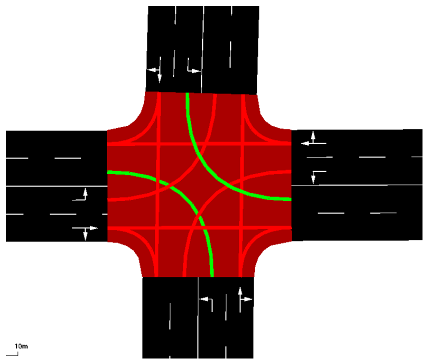

Fig. 5. Example of a "left phase" in SUMO.

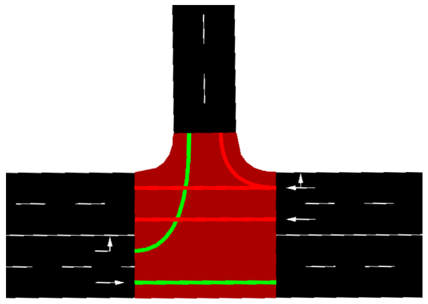
Fig. 6. Example of a "straight left phase" in SUMO.

For the set Obs_{int} meanwhile, an additional 39 predicates were instantiated for use in the $cond$ of rules in RS_{int}. Unlike the predicates used in rules in RS, these can describe communicated intentions from other agents and the time since a given communication was received. The predicates describing the time since a communication was received take on similar range descriptions as those mentioned above. Rules for the sets RS^{exp} and RS^{exp}_{int} meanwhile are defined during experimental evaluation. We have however instantiated four predicates to be used by rules in this set, two of which specify which direction an emergency vehicle is approaching from. The other two check if a particular phase in an intersection has spent the maximum time in a green or yellow state. For learning, the maximum length for a green phase is 225 seconds, and the maximum yellow phase length is 5 seconds. These two predicates are intended to keep intersections from getting stuck in a particular phase.

We also needed to define the set of actions Act, to be applied when a rule is selected by an intersection controller agent. Traffic lights in our study will adhere to the same signal phase changing conventions Canadian traffic lights do, and so possible actions are bounded by said conventions, too. As such, actions when a given rule is selected to be applied by an agent include: (1) transitioning from a red phase to a green phase, (2) transitioning from a green phase to yellow phase, or (3) do-nothing. It may appear odd to omit the action of transitioning from yellow to red, but SUMO ignores explicitly describing this action, since it is implicit when a different direction is granted a green phase. The contents of Act will vary from agent to agent, depending on their intersection configuration, which SUMO defines automatically, and the system cycles through it as it sees appropriate. The value of p_{coop} in our experiments is 0.5, meaning the agent will choose an action from a shout-ahead rule half of the time, while the value of the exploration vs exploitation parameter ϵ described in [5] is the same, i.e 50% of the time, the agent will exploit one of the rules with the highest weight, and the other 50% of the time it does exploration.

B. Instantiating the hybrid behaviour learning

The hybrid behavior learning method for the Shout-ahead architecture is responsible for optimizing the agent's behavior. As mentioned prior, two learning algorithms are employed by this method. The main one is an evolutionary algorithm, creating new individuals using the crossover and mutation operators, with respect paid to rule weights in selecting individuals to be used in these operators, and some additional random factors. Given the aforementioned variance in action sets, depending on intersection configuration, we instantiate an agent pool AP per action set type, meaning agents must share identical intersections to share an agent pool. As such, APS varies in size depending on the road network configuration. Additionally, rule sets RS and RS_{int} are at first instantiated with some number of random rules. We also naturally need a fitness measure fit for an individual (ind), which we define as:

$$fit(ind) = sum(rfitValuesAccrued)/numOfSimRuns$$

where $rfitValuesAccrued$ represents a list of all the $rfit$ values achieved by the individual in its lifetime and $numOfSimRuns$ is the number of simulation runs the individual has participated in. If the individual has not participated in any simulation runs, $fit = 10,000$. An individual's run fitness $rfit$ meanwhile, is generated at the completion of every simulation run using the following formula:

$$rfit(ind) = \begin{cases} simTime - sumoRT & if\ simTime < sumoRT \\ +fitpen & \\ indivAVT & elif\ indivAVT = bestAVT \\ +fitpen & \\ indivAVT * 10 & elif\ indivAVT < 1.1 * bestAVT \\ +fitpen & \\ indivAVT * 20 & elif\ indivAVT < 1.2 * bestAVT \\ +fitpen & \\ indivAVT * 30 & elif\ indivAVT < 1.3 * bestAVT \\ +fitpen & \\ indivAVT * 40 & else \\ +fitpen & \end{cases}$$

where ind is the individual being evaluated, $simTime$, $simTime \geqslant 0$, is the runtime of the simulation the individual just participated in, $sumoRT$ is the runtime of SUMO's Actuated Traffic Light algorithm for the traffic network and vehicle flow being used, $indivAVT$, $indivAVT \geqslant 0$, is the aggregate wait times sustained by vehicles at the intersection the individual is assigned to, and $bestAVT$, $bestAVT \geqslant 0$, is the lowest $indivAVT$ value in the agent pool the individual belongs to. $fitpen$ is a sum of penalty constants incurred every time an individual fails to apply an action from one of its own rules, or the action applied from its rule results in a worse traffic state in the individual's assigned intersection than before the application of the rule. In order to ensure a traffic light does not get stuck in a given phase in perpetuity, maximum phase lengths for green and yellow phases are applied, and an individual that relies solely (or mostly) on these for phase switching can achieve a significantly lower runtime than the ATL algorithm. $fitpen$ is intended to reprimand strategies that do this and ensure the learner doesn't select for strategies that do not apply any rules. This formula prefers individuals who incur smaller runtimes than SUMO's Actuated Traffic Light (ATL) algorithm, seeing as the objective of the system is to produce strategies that are more efficient than currently used traffic signal controller algorithms. Should a strategy run longer than the ATL algorithm, the fitness is instead calculated based on how much longer vehicles waited at an individual's intersection compared to the lowest aggregate wait time in the agent pool. This is done to lead the learner to select for strategies that minimize vehicle wait times. Our evolutionary learner selects for minimal fitness values, and runs for 50 generations with the objective of generating individuals with the lowest possible (negative) fitness value. A smaller penalty is also incurred if an individual does apply an action from one of its rules but said application worsens the state of the intersection. That is, if the number the vehicles waiting to proceed through an intersection is larger after the application a rule, that rule is considered to have worsened the state of the intersection. The $fitpen$ value is then cumulatively added to the individual's $rfit$ value for the given simulation run. For our learning, $fitpen = 30$ if no rule is applicable at a given time step, and $fitpen = 10$ if a rule's action is applied but worsens the state of the intersection.

The other learning algorithm employed is the aforementioned SARSA variant of reinforcement learning, which agents perform on the weights of their rules in their rule sets [8]. Rule weights are updated according to the method described in Section II, with one adjustment. Our reward r is defined based on two inputs, $throughputRatio$ and $waitTimeReducedRatio$, calculated using the following formula:

$$r(ind) = \quad throughputRatio * \kappa + waitTimeReducedRatio \\ * \iota + \ rulepen$$

where κ, $0 \leqslant \kappa \leqslant 1$, is the $throughput$ factor and ι, $0 \leqslant \iota \leqslant 1$, is the $waitTimeReducedRatio$ factor. Both $throughputRatio$ and $waitTimeReducedRatio$ are normalized ratios representing the relative impact on the state of the intersection. The value for $throughputRatio$ is calculated using the ratio of actual throughput to the total amount of cars in the intersection, while $waitTimeReducedRatio$ is derived from the ratio of waiting times accrued by the cars that passed through the intersection to the sum of all waiting times at the intersection before the rule was applied. A penalty $rulepen$ is applied alongside the reward of rules whose application results in an intersection being in a worse state than before the rule's application. This follows the same principle for evaluating intersection states used for $fitpen$. $rulepen$ is calculated as $intersectionQueueDifference * \rho$, where $intersectionQueueDifference$ is the difference between the number of vehicles waiting at the intersection after the rule's action is applied and the number of vehicles waiting before it, and ρ, -1 $\leqslant \rho \leqslant$ 0, is a multiplier parameter. For our experiments, ρ = -0.05.

These two learning phases each play crucial roles in improving the performance of the intersection controller agents, with the evolutionary algorithm providing new individuals that could potentially perform better than existing ones, and the reinforcement algorithm painting a better picture of what rules could be used to create these better individuals.

IV. Experimental Evaluation of Proposed Solution

In this section, we will lay out our experimental evaluation of the solution as outlined in Section III. We will first describe the general set-up of the experiments, then analyze the performance of learning and shout-ahead integrated traffic signal controller agents relative to a default "control" algorithm mimicking currently used traffic signal timing algorithms provided by SUMO, and finally compare our approach using two sets of rules with using the additional user-defined rule sets described in Section III (i.e using four rule sets).

A. Set-up

Given the purpose of this project is to demonstrate that a traffic signal controller integrated with learning and the shout-ahead architecture is superior to currently available traffic signal control algorithms, we must naturally pit the two against each other in evaluations. Serving as our currently in-use traffic algorithm is SUMO's Actuated Traffic Light (ATL) algorithm, which we presented in Section II [8]. We use all of the default parameter values for this algorithm, including the maximum time gap between vehicles causing a phase to be prolonged, which is three seconds [8].

For learning, to give the evolutionary learning component of the system time to learn effective strategies, we ran our system for 50 generations. We set $|RS|, |RS_{int}| = 10$, with $1 \leqslant |cond| \leqslant 3$ for their rules. We require each individual to participate in a minimum of 3 simulation runs (with 3 different teams) per generation, and form learned teams with the best individuals from each agent pool upon the conclusion of learning. In theory, these individuals should be the most proficient at managing their own intersections and handling the intentions of their neighbours through shout-ahead, thus forming the optimal learned team. Given the time constraints for testing our system, we chose to learn and test on a simple SUMO network. The most simplistic road network for our system must at minimum incorporate one of each type of intersection described above: a standard 4-arm intersection, a T-intersection and a 4-arm Incoming Layout intersection, such as the one in Fig. 7. In SUMO, intersections are represented by

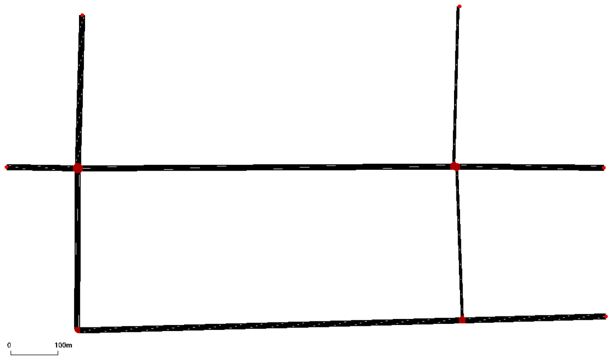

Fig. 7. Example of a simple network in SUMO.

the relatively large, red hexagons or octagons where different roads meet, while the red dots represent boundary points of the network, where vehicles originate and exit the network from. Any other red markers serve as SUMO's representation of bends in a road. Our chosen network results in $|APS| = 3$, and we instantiated 30 individuals in each agent pool to provide a sufficient gene pool for the evolutionary learner.

Finally, we generated 3 random, unique traffic flows to be used for both learning and testing. The flows naturally range in vehicle quantity so to evaluate the system across different demand levels. The lightest flow contains 75 vehicles while the heaviest contains 425 vehicles. To third flow serves as a compromise between the first two, containing 225 vehicles, allowing us to evaluate the system on a wide range of flow sizes. All experiments were performed on a Windows-based ASUS ROG Strix dekstop, with an Intel Core i7 processor running at 3.6GHz, and 8.00 GB of RAM.

Lastly, before we learn we must determine $sumoRT$ for the three different flows we are using. For the 75 vehicle flow, SUMO's ATL algorithm records a runtime of 1264 seconds, therefore $sumoRT = 1264$. For the 225 vehicle flow meanwhile, SUMO's ATL algorithm records a runtime of 1690 seconds, therefore $sumoRT = 1690$. Finally, the 425 vehicle flow, SUMO's ATL algorithm records a runtime of 2837 seconds, therefore $sumoRT = 2837$.

B. Results without user-defined rule sets

As mentioned above, upon the completion of learning we form a team with the best strategies from each of the agent pools for the purpose of testing. In our first set of experiments, we evaluate each of the learned strategy teams on the same network and with the vehicle flow they learned with to see if our shout-ahead and learning implemented agents are superior to SUMO's ATL algorithm.

TABLE I
COMPARISON BETWEEN PERFORMANCE OF AGENTS WITH THE SUMO ATL STRATEGY VS. THE LEARNED SHOUT STRATEGY ON THE VEHICLE NETWORK AND FLOW USED FOR LEARNING

Vehicle Flow	SUMO ATL strategy runtime (seconds)	Learned shout-ahead strategy runtime (seconds) [20 sim avg]
75 vehicles	1265	DNF
225 vehicles	1690	1485
425 vehicles	2837	1939
Average	1931	N/A
Average without 75 vehicle flow	2264	1712

TABLE II
COMPARISON BETWEEN PERFORMANCE OF AGENTS WITH THE SUMO ATP STRATEGY VS. THE LEARNED SHOUT STRATEGY WITH VARIOUS DEPARTURE TIME CHANGES IN THE 225 VEHICLE FLOW

Vehicle Flow Change	SUMO ATL strategy runtime (seconds)	Learned shout-ahead strategy runtime (seconds) [20 sim avg]
10%	1406	1257
20%	1406	1259
30%	1406	1258
40%	1406	1259
50%	1406	1258
Average	1406	1258

It is obvious from the results in Table I that our shout-ahead and learning enabled agents as currently configured are not always superior to SUMO's default ATL algorithm. With a smaller 75 vehicle flow, the shout-ahead and learning strategy was unable to finish the simulation, stranding 16 vehicles at the 4-arm intersection. This is because the learned strategy for this intersection does not contain a rule with an action that grants a green phase to the direction of the stranded vehicles. We use the same predicate set for learning all three flows, which may have been a disservice to the smaller flow. For instance, the predicate sets $numCarsWaitingToProceedStraight$ and $numCarsWaitingToTurnLeft$ contain range sub-predicates to bin the number of vehicles waiting at an intersection. $numCarsWaitingToProceedStraight$ sub-predicates have 5 vehicle ranges (i.e $numCarsWaitingToProceedStraight_{05}$ evaluates to true if between 0 and 5 inclusive vehicles are at the intersection) until 15 vehicles then switch to 10 vehicle ranges, going up to 45 vehicles. Since the vehicles are introduced into the network one at a time, it's highly unlikely any one intersection would have anywhere near 45 vehicles waiting at it at one time, let alone all of them intending to proceed straight. These potentially inapplicable predicates would have circulated through the rules of strategies as learning progressed, hampering the potential of the strategy from the moment it enters one of its rules. These larger predicate ranges are likely far better suited for the bigger vehicle flows, as displayed in the successes of the learned strategies on the two bigger flows.

We were also interested in what effect the structure of the vehicle flow had on the performance of the agents. That is, how much of an impact do the vehicle departure schedules have on the agent performance? Naturally, our learned shout-ahead strategies have a narrow use-case: they're only relevant for the specific flow they were trained on. But will changing the departure times of certain vehicles, thus also changing when high demand times occur, affect the performance of the agent, or will its learned rules handle it the same? We alter the flow five different times, increasing the percentage of departure times changed by 10% each time (number rounded down if it results in a decimal), starting at a 10% change in departure times. We do not change or exceed the time the last vehicle departs so to not affect the runtime in that manner. All times are randomly adjusted within 10 seconds of their original value. The results of these tests can be seen in Table II. Both SUMO's default ATL algorithm and our learned strategies benefit from the incremental changes in the 225 vehicle flow, and both also perform consistently through all 5 changes. The performance increase for the learned strategy is an especially surprising outcome, given by the fifth changed flow, half the vehicles from its training flow had different departure times. This may be because we maintained the number of vehicles passing through each intersection during the simulation, and the changes in departure times created more favourable conditions for the rules of the learned strategies.

C. Results with user-defined rule sets

We are also interested in the effect that user-defined rules have on the performance of our strategies. The strategies did not learn with the user-defined rules, but they are instead included into RS^{new} and RS^{new}_{int} of the learned strategies immediately before a test simulation. We decided to included rules that police the maximum phase lengths for green, yellow and red phases, as well as one that ensures a direction with vehicles waiting to receive a green phase if all other directions are empty. These rules are applied both separately and together, as described below.

We first added user-defined rules for the maximum phase length of both green and yellow phases. SUMO natively allows for maximum phase lengths to be set for each individual phase of a traffic light, and during learning we set the maximum green phase length to be 225 seconds and the maximum yellow phase length to be 5 seconds. Our rules overwrite the values used by SUMO, splitting them in half (rounded to nearest whole number) such that the maximum green phase length is 113 seconds and the maximum yellow phase length is 3 seconds.

The results in Table II paint an important picture about how user-defined rules are not guaranteed performance boosts. By reducing the maximum green and yellow phase lengths, it is likely the learned strategies for the 225 vehicle flow were

TABLE III
COMPARISON BETWEEN PERFORMANCE OF AGENTS WITH THE SUMO ATP STRATEGY VS. THE LEARNED SHOUT STRATEGY WITH A USER-DEFINED RULE FOR MAXIMUM GREEN AND YELLOW PHASE LENGTHS

Vehicle Flow	SUMO ATL strategy runtime (seconds)	Learned shout-ahead strategy runtime (seconds) [20 sim avg]
75 vehicles	1265	DNF
225 vehicles	1690	2847
425 vehicles	2837	1939
Average	1931	N/A
Average without 75 vehicle flow	2264	2393

TABLE IV
COMPARISON BETWEEN PERFORMANCE OF AGENTS WITH THE SUMO ATP STRATEGY VS. THE LEARNED SHOUT STRATEGY WITH A USER-DEFINED RULE FOR MAXIMUM RED PHASE LENGTH

Vehicle Flow	SUMO ATL strategy runtime (seconds)	Learned shout-ahead strategy runtime (seconds) [20 sim avg]
75 vehicles	1265	954
225 vehicles	1690	2785
425 vehicles	2837	3075
Average	1931	2271

TABLE V
COMPARISON BETWEEN PERFORMANCE OF AGENTS WITH THE SUMO ATP STRATEGY VS. THE LEARNED SHOUT STRATEGY WITH USER-DEFINED RULES FOR MAXIMUM GREEN, YELLOW AND RED PHASE LENGTHS

Vehicle Flow	SUMO ATL strategy runtime (seconds)	Learned shout-ahead strategy runtime (seconds) [20 sim avg]
75 vehicles	1265 seconds	836
225 vehicles	1690 seconds	2847
425 vehicles	2837 seconds	4822
Average	1931	2835

TABLE VI
COMPARISON BETWEEN PERFORMANCE OF AGENTS WITH THE SUMO ATP STRATEGY VS. THE LEARNED SHOUT STRATEGY WITH A USER-DEFINED RULE FOR GRANTING A GREEN PHASE TO A DIRECTION WITH VEHICLES WAITING WHILE ALL OTHER DIRECTIONS ARE EMPTY

Vehicle Flow	SUMO ATL strategy runtime (seconds)	Learned shout-ahead strategy runtime (seconds) [20 sim avg]
75 vehicles	1265	718
225 vehicles	1690	1179
425 vehicles	2837	1346
Average	1931	1081

tripped up. Their rules were tuned for a certain maximum green and yellow phase length, and they learned to perform within those boundaries. By changing the boundaries, their rulesets were no longer operating in their optimal environment and performed worse. The learned strategies for the 75 vehicle flow still stranded vehicles and therefore did not finish. This is unsurprising since vehicles were left stranded when using the same values for maximum green and yellow phase length as we did during learning.

While SUMO allows for maximum phase lengths to be specified, it only allows for the maximum lengths of green and yellow phases to be declared (since red phases are simply a result of not being in a green or yellow phase). In Table III we see the results of our next experiment, where we leave the maximum green and yellow phase lengths as they were during learning, and introduce a user-defined rule that is applied if a phase remains red for a certain amount of time. We define a constant $MaxRedPhaseTime$ = $numberOfGreenPhases$ - 1 * $maxGreenPhaseTime$ + $numberOfYellowPhases$ - 1 * $maxYellowPhaseTime$ for each traffic light, and the condition of this rule evaluates to true if any of a traffic light's phases remains red for longer than that constant value. That number is equal to the amount of time it would take for every other phase to spend the maximum amount of time in both their green and yellow phases.

The results from Table IV further highlight the fact that user-defined rules must be carefully designed. While the inclusion of the maximum red phase length rule turned our learned strategy for the 75 vehicle flow from unusable to excellent, with a runtime reduction of nearly 25% over SUMO's ATL strategy, the learned strategy for the 225 vehicle flow was again hampered by it and the learned strategy for 425 vehicles was downright torpedoed. We discussed previously that the learned strategies for the 75 vehicle flow had non-robust rulesets, without a rule action for every phase in the intersections they were serving. The maximum red light length will have made it so that certain directions that would not have otherwise received a green phase, did so, therefore avoiding the stranding that occurred without it in place and turning the learned strategy itself from a complete failure to resounding success. However, for the two bigger flows it is likely the rules from the learned strategies were overruled by the new user-defined rule at various points in the simulation, creating less favourable conditions than if the learned strategy's rule had been applied.

We then added back our user-defined rules for the maximum phase length of both green and yellow phases alongside our user-defined rule for the maximum red phase length, with the results of this experiment displayed in Table V. The results are much in line with the results from Table III and Table IV, where the user-defined rules for maximum green and yellow phase length and maximum red phase length were used separately. The 75 vehicle flow saw a massive increase in performance from the maximum red phase length user-defined rule's lone inclusion, so it follows that when combining it with new, lower maximum green and yellow phase lengths (which are used to define the maximum red phase length, as described

TABLE VII
COMPARISON BETWEEN PERFORMANCE OF AGENTS WITH THE SUMO ATP STRATEGY VS. THE LEARNED SHOUT STRATEGY WITH ALL 4 USER-DEFINED RULES

Vehicle Flow	SUMO ATL strategy runtime (seconds)	Learned shout-ahead strategy runtime (seconds) [20 sim avg]
75 vehicles	1265	504
225 vehicles	1690	1136
425 vehicles	2837	1992
Average	1931	1211

above), the runtime will increase. On the flip side, the learned strategies for the 225 and 425 vehicle flows were hampered by the user-defined rule for maximum red phase length, so it follows that they would be further hurt by an even more restrictive version of that rule during these tests.

Next, we removed the three previously described rules from RS^{new} and RS^{new}_{int} and replaced them with a rule with a condition that evaluates to true if there vehicles waiting to proceed through the intersection in one direction, and all other directions have no vehicles waiting. The rule's action grants a green phase to the direction with the waiting vehicles. The results of this test are in Table VI, and mark the first time the average of the 3 learned strategy's runtimes is lower than that of SUMO ATL strategy. This is likely because this user-defined rule targets inefficiencies in the learned strategies rule-sets, providing green phases and expediting vehicles waiting at intersections providing green phases to directions with empty vehicle queues. While these vehicles would've eventually been serviced by a rule from the learned strategies, this user-defined rule ensures the process happens immediately, pushing the simulation along at a faster rate.

Finally, we applied all 4 user-defined rules at once, with the results of this experiment in Table VII. Knowing the individual effects of the user-defined rules on the different flows, the combined effect results in, predictably, a blend of the individual ones. The 75 vehicle flow benefited to various levels from the individually applied user-defined rules (aside from the maximum green and yellow phase length), and so all of them applied in concert resulted in its lowest runtime yet. The 225 only benefited from the last user-defined rule we tested, but its benefits were so extreme it outweighed the negative effects of the others. On the other hand, the negative effects of the maximum green, yellow and red phase length user-defined rules were so significant when applied individually to the learned strategy for the 425 vehicle flow, they outweighed the positive effects of the last user-defined rule. This highlights that user-defined rules must be designed for specific flows, as they are not a universal performance increase. In fact, ineffective user-defined rules can actually harm runtime significantly.

V. Related Works

Naturally, there has been a lot of work in the area of traffic signal optimization with artificial intelligence and machine learning. Most traffic light controllers rely on pre-timed cycle policies, but [9] showed that using an intelligent agent to generate traffic signal controller (TSC) policies was more effective. Similar findings were presented in [10], where agents learned a near-optimal policy. Furthermore, [11] highlighted that many current solutions to the TSC problem are centralized, and argued that this centralized nature makes them infeasible for large scale adaptive traffic signal control due to the high dimension of the joint action space. As such, decentralized solutions must be explored in order to find an optimal solution to TSC problem, a characteristic shared by this project and its related works. While specific methods vary between the papers, their end-goals align: to optimize traffic throughput through the implementation of algorithms that utilize artificial intelligence and machine learning principles.

The use of multi-agent reinforcement learning (MARL) was the most popular method observed, solely utilized by [12], [13], [10], [14], [15], [16], [17], [18], [19], [20] and [21]. Naturally, nuances exist between each of the studies, despite their shared solution to the TSC problem. [12] constructed the problem as a discounted cost Markov decision process, and applied multi-agent Q-learning to obtain dynamic traffic signal control policies, with Q-factors updated based on communication with neighboring TSC agents. [13] and [19] make use of similar neighbor communication approaches, except [13] utilizes the max-plus algorithm to achieve coordination between TSC agents, while [19] defines their own coordination algorithm. In both cases, agent protocols are improved via reinforcement learning. Unlike [12] and [17], [13] applies its solution directly to a large-scale problem to verify its efficacy in realistic settings, and found success with this approach.

While communication and cooperation between agents seems intuitively pragmatic, it can be difficult to discern what information is useful to share and what the consequences of such cooperation are. [21] investigates this very topic in the context of a distributed urban traffic control, where each junction is controlled by an independent agent. The authors analyze both the benefits of sharing information, and the consequences of doing so.

Naturally, it must be acknowledged the entire premise of this project is rooted in the work presented in [5], which lays out the shout-ahead architecture and hybrid behavior learning method in abstracted detail, creating a blueprint for future implementations to follow. Though the application area of [5] does not align with this project, the success of its experimental evaluations merit the implementation of the architecture in a different application area.

VI. Conclusion and future work

Ineffective traffic signal timing algorithms can exacerbate poor traffic conditions, creating bottlenecks leading to traffic jams. Traffic signals within individual intersections are managed by a controller agent, which is responsible for changing signal states based on an algorithm that relies on a predetermined signal timing cycle. Though the cycle can be altered based on heavier traffic flow, as is seen with SUMO's Actuated Traffic Light algorithm, applicable adjustments are also predetermined and thus don't take into consideration the full state of the intersection, information that could lead to a different action being taken. By implementing the Shout-ahead agent architecture and a hybrid behavior learning method for it into intersection controller agents, these agents gain a level of dynamic flexibility in their decision making, allowing them to take the most optimal action for the current state of their environment, instead of being bound to rigid timing-cycle. The ability for a traffic signal control agent to learn which actions are proper in which situations, coupled with communication with its neighboring cohorts, results in more optimal actions being taken, and more effective intersection control than a near-static algorithm would provide.

There are many potential directions for future work. The most obvious would naturally be testing the system in more complex networks with heavier and more varied vehicle flows. While its performance on a simple network serves as a proof of concept for the idea of optimizing traffic signal controllers with learning and the shout-ahead architecture, it still does not allow us to make any statements regarding its general effectiveness. In a similar vein, integrating the ability to learn multiple rule sets for different flows within the same strategy (individual) would greatly expand the versatility of the system and increase its viability for real world application. For instance, the ability to learn separate strategies for morning and evening rush hour, non-rush hour day and nighttime flows would align the system's capabilities with that of currently used algorithms [4].

Our system also contains a vast parameter set, offering a myriad of parameter value combinations, including but not limited to the number of generations the system takes to learn, the number individuals per agent pool, the number of simulation runs each individual must complete per generation, and the number of rules in the rule sets of each individual. The fitness, reward and various penalty functions also take in inputs that are weighed relative to each other based on factor parameters. One parameter adjustment can have a rippling effect across the entire system, thus determining good parameter values would be valuable.

Another interesting direction for future work concerns the user-defined rule sets RS^{exp} and RS^{exp}_{int}, and how different utilizations of them affect the performance of the system, specifically learning with them enabled (which we did not do). These rule sets provide a unique opportunity to integrate various knowledge into the system, and the effects of doing so could drastically alter the performance and behaviour of the system, as we saw during our experimental evaluation.

References

[1] M. Bennardo, "Statscan study shows canadian commute times are getting longer — and it's costing us," https://www.cbc.ca/news/business/statistics-canada-commute-times-study-1.5038796, 2019, accessed: Sept. 21, 2019.

[2] J. G. T. Yaropud and S. LaRochelle-Côté, ""results from the 2016 census: Long commutes to work by car," https://www.cbc.ca/news/business/statistics-canada-commute-times-study-1.5038796, 2019, accessed: Sept. 21, 2019.

[3] X. Z. Y. Shaoxin and Y. An, "Identification and optimization of traffic bottleneck with signal timing," *Journal of Traffic and Transportation Engineering (English Edition)*, vol. 2, pp. 353–361, October 2014.

[4] E. National Academies of Sciences and Medicine, *Signal Timing Manual – Second Edition*. Washington, DC: The National Academies Press, 2015.

[5] J. D. S. Paskaradevan and D. Wehr, "Learning cooperative behaviour for the shout-ahead architecture," *Web Intelligence and Agent Systems: An Internation Journal*, vol. 12, pp. 309–324, Oct 2014.

[6] R. Sutton and A. Barto, *Reinforcement Learning: An Introdution*. The MIT Press, 1998.

[7] J. E. M. Behrisch, L. Bieker and D. Krajzewicz, "Sumo–simulation of urban mobility: an overview," in *Proc. Third International Conference on Advances in System Simulation (SIMUL 2011)*, Barcelona, Spain, Oct 2011.

[8] "Automatically generated tls-programs," simulation/traffic lights – sumo documentation," https://sumo.dlr.de/docs/Simulation/Traffic_Lights.html#actuated_traffic_lights, 2020, accessed: Jan. 17, 2020.

[9] N. M. M. Abdoos and A. L. C. Bazzan, "Traffic light control in non-stationary environments based on multi agent q-learning," in *Proc. 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Washington, DC, Oct 2011.

[10] J. L. P. Gregoire, C. Desjardins and B. Chaib-draa, "Urban traffic control based on learning agents," in *Proc. 2007 IEEE Intelligent Transportation Systems Conference*, Seattle, WA, Oct 2007.

[11] B. d. S. E. B. D. de Oliveira Boschetti, A. Bazzan and L. Nunes, "Reinforcement learning based control of traffic lights in non-stationary environments: A case study in a microscopic simulator," in *Proc. 4th European Workshop on Multi-Agent Systems EUMAS'06)*, vol. 223, Lisbon, Portugal, Dec 2006.

[12] A. N. H. K. K. J. Prabuchandran and S. Bhatnagar, "Multi-agent reinforcement learning for traffic signal control," in *Proc. 17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Qingdao, China, Oct 2014.

[13] B. B. L. Kuyer, S. Whiteson and N. Vlassis, *Multiagent Reinforcement Learning for Urban Traffic Control Using Coordination Graphs*, ser. Lecture Notes in Computer Science. Springer Science + Business Media, 2008, vol. 5212, pp. 656–671.

[14] M. Wiering, "Multi-agent reinforcement learning for traffic light control," in *Proc. Seventeenth International Conference on Machine Learning (ICML 2000)*, Stanford, CA, Jun 2000.

[15] M. A. Khamis and W. Gomaa, "Enhanced multiagent multi-objective reinforcement learning for urban traffic light control," in *Proc. 2012 11th International Conference on Machine Learning and Applications, ICMLA 2012*, Boca Raton, FL, Dec 2012, pp. 586–591.

[16] T. U. I. Arel, C. Liu and A. G. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control," *IET Intelligent Transport Systems*, vol. 4, pp. 128–135, 2010.

[17] E. V. der Pol and F. Oliehoek, "Coordinated deep reinforcement learners for traffic light control," in *NIPS'16 Workshop on Learning, Inference and Control of Multi-Agent Systems*, Barcelona, Spain, Dec 2016, pp. 586–591.

[18] L. Z. D. Houli and Z. Yi, "Multiobjective reinforcement learning for traffic signal control using vehicular ad hoc network," *EURASIP Journal on Advances in Signal Processing*, vol. 2010, 01 2010.

[19] S. El-Tantawy and B. Abdulhai, "Multi-agent reinforcement learning for integrated network of adaptive traffic signal controllers (marlin-atsc)," 09 2012, pp. 319–326.

[20] L. K. B. Bakker, S. Whiteson and F. Groen, *Traffic Light Control by Multiagent Reinforcement Learning Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 475–510.

[21] D. Oliveira and A. Bazzan, *Multiagent Learning on Traffic Lights Control: Effects of Using Shared Information*. Information Science Reference, 2009, pp. 307–321.