Interim report: 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 n 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. Our proposed solution to this problem involves implementing a shout-ahead agent architecture and a hybrid learning behavior method into traffic signal controller agents to better optimize their performance. The performance of the shout-ahead enabled traffic signal control agents will be determined over a set of experimental evaluations, using simulation time and traffic throughput as performance measures, and will compare the shout-ahead traffic signal controller's performance relative to that of the standard signal control algorithm utilized by SUMO to determine the effectiveness of our implementation.**

Keywords—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 a 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 Ag is represented as a quadruple $Ag = (Sit, Act, Dat, f_{Ag})$, where Sit is the set of situations Ag can distinguish, Act is the set of actions Ag can perform, Dat is the set of possible value combinations of Ag's internal data areas (all internal "states" Ag can have) and $f_{Ag} : Sit \times Dat \rightarrow Act$ is Ag'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 *Env*, such that $A = \{Ag, Ag_1, \ldots, Ag_n\}$ with $Ag = (Sit, Act, Dat, f_{Ag})$, $Ag_i = (Sit_i, Act_i, Dat_i, f_{Agi})$ for all i, $1 \le i \le n$. At the core of the shout-ahead architecture is the concept of a *rule with weight*, which holds the general form:

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

Naturally, $a \in Act$ (Act will always contain a do-nothing action). The condition $cond$ must 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 $cond = \{obs_i, \ldots, obs_m\}$ 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 $f_{eval} : 2^{Obs} \times Sit \times Dat \rightarrow Boolean$. Subset $Obs_{int} \subseteq Obs$ 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 Ag_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 $Obs \setminus Obs_{int}$ 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 Ag'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} actually consists of sub-functions, f^{int}_{Ag} and f^{fin}_{Ag}. f^{int}_{Ag} 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^{fin}_{Ag} works the same way, except it operates on RS_{int} instead. Then, Ag 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. 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. As mentioned prior, an agent Ag belongs to a pool, which itself represents the population of agent strategies for Ag. 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 $APS = (AP, AP_1, \ldots, AP_k)$ of agent pools such that AP contains possible strategies (individuals) for Ag and for each $Ag_i \in A$, there is an agent pool AP_j such that AP_j contains possible strategies for Ag_i. For a scenario $Scen$, we have as *team requirement* TR a certain number of agent pools, i.e $TR = (AP'_1, \ldots, AP'_l)$, with $AP'_j \in \{AP, AP_1, \ldots, AP_k\}$. A simulation team $st = (ag_1, \ldots, ag_l)$ 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 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 [8]). 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:

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

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 Ag, 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 $(RS^{new}, RS^{new}_{int})$ 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^{new}_{int}. If RS^{new} or RS^{new}_{int} 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 by applying the strategy in several simulation runs $sr_1, \ldots sr_{smax}$ with different simulation teams, combining the $rfit(sr_i, ag)$-values.

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. 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.

III. Proposed Solution to the problem

In this section, we present my 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 completely different game and action states, 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 intersection, 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. Unlike the other two, this intersection type is more complex, and potentially presents more bottleneck, 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. These two rule sets contain user defined rules, one for RS and one for RS_{int}. Rules within this set have maximum priority, such that if a rule within this set is applicable at a 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 many individual strategies ("Individuals") will exist per agent pool, how many rules can exist in one rule set and how many observations can make up the condition of a rule itself. As such, our preliminary decisions on these matters are as follows: agent pools will contain 5 individuals each, with 10 rules per individual.

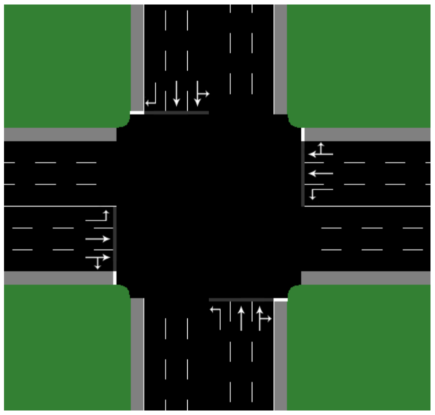
Fig. 1. Example of a Standard 4-arm Intersection in SUMO.

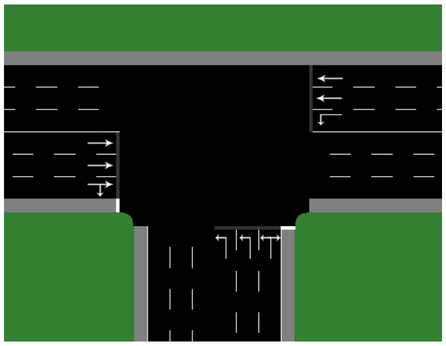
Fig. 2. Example of a T- Intersection in SUMO.

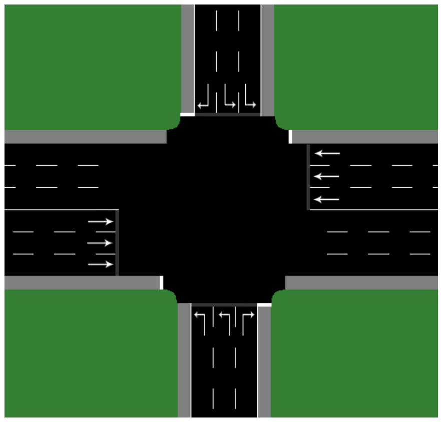
Fig. 3. Example of a 4-arm Incoming Layout Intersection in SUMO.

Every rule can have up to 3 observation predicates make up its condition. We developed a set of 81 observation predicates to be used 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 our environment, 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) NumOfCarsWaitingToProceedStraight, 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.

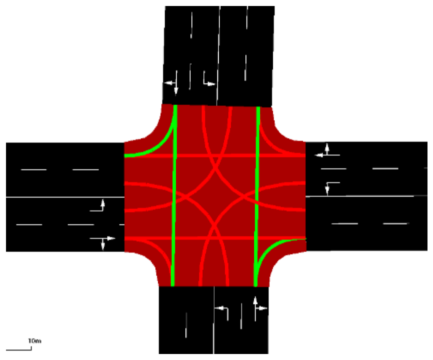

Fig. 4. Example of a "straight phase" in SUMO.

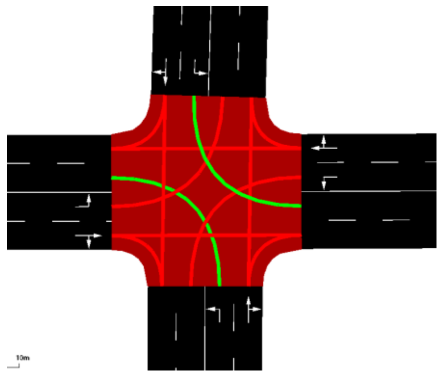

Fig. 5. Example of a "left phase" in SUMO.

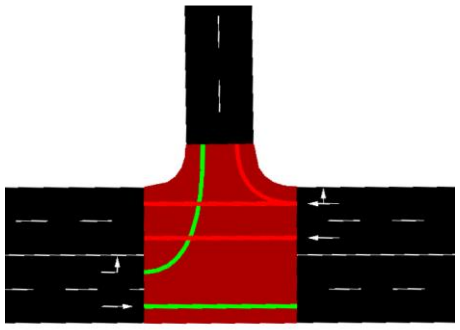

Fig. 6. Example of a "straight left" phase in SUMO.

A traffic light state 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. The set RS_{int} meanwhile has its own set of 39 predicates it uses to create rules. Unlike the predicates used 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. The design of these predicates greatly affects how the agent behaves, and they will likely be adjusted as the evaluation process evolves to account for circumstances unforeseen at this point in time. The user-defined rule set for *RS* contains 4 predicates, two of which specify which direction an emergency vehicle is approaching from. The other two check if an intersection has spent the maximum time in a green or yellow state. These two predicates are intended to keep intersections from getting stuck in a particular phase. The user-defined rules for RS_{int} meanwhile are defined from a text file that the system inputs and uses. This allows the user to easily define whatever rules they desire before runtime. This strategy could also be implemented for user-defined rules in *RS*, and a decision on the matter will take place in testing.

We also defined a set of actions 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. Actual action sets vary from agent to agent, depending on their intersection configuration, which SUMO defines automatically, and the system cycles through it as it sees appropriate.

B. Instantiating the hybrid behaviour learning

The hybrid behavior learning method for the Shout-ahead architecture is then 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. Naturally, we need a fitness measure for an individual, which would be determined by doing simulation (training) runs. An individual's fitness is calculated by the following equation:

$$\tau * (1/\mathit{simulationTime}) + \delta * (1 - 1/\mathit{sumOfAllRuleWeights})$$

where τ, $0 \leq \tau \leq 1$, is the simulation time weight factor and δ, $0 \leq \delta \leq 1$, is the rule weight factor. This equation rewards the individual with the fastest simulation time and highest aggregate rule weights. The weights of these two factors relative to each other can be adjusted by the user, and optimal values will be sought in testing. For the time being, both factors are set to 1, giving them equal weight to each other. The other learning algorithm employed is a SARSA variant of reinforcement learning, which agents perform on the weighs of their rules in their rule sets [7]. When selecting a rule by the process described in Section II, they give a higher priority rules that have yielded positive results in the past and lower priority to the opposite. 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 better individuals. Altogether, if the objective is to increase traffic throughput, over time it is expected that individuals will be created that operate more efficiently than current intersection controller algorithms.

IV. Intended Evaluation of Proposed Solution

In this section, we will lay out my proposed experimental evaluation of the solution outlined in Section III. 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.

SUMO provides a few default signal-timing algorithms that are similar to the preset ones used in intersections today [4][8]. One such algorithm 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. Another signal-timing algorithm built into SUMO involves “actuated traffic lights”. This algorithm supports gap-based actuated traffic control, which prolongs traffic phases when a continuous vehicle stream is detected in the intersection. 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, making it an attractive candidate to evaluate the learning and shout-ahead equipped agents in testing [4]. Additionally, SUMO allows for the importing of other signal-timing algorithms, however they are likely to require some retrofitting to prepare them for the SUMO interface, an effort proposition that greatly lessens its viability for our current purposes [8].

In addition to testing the learning and shout-ahead integrated traffic signal control agents against a “control” algorithm discussed above, it would be equally interesting to compare its performance against agents with the shout-ahead component turned off, leaving only learning enabled. Precisely, this would mean making use of only one rule set *RS* instead of two, *RS* and RS_{int}. Doing so would provide us insight into the benefits and overall effectiveness of the shout-ahead architecture in this application area. Experimental evaluation in [5], in the context of the Battle for Wesnoth, showed agents using shout-ahead were 2.5 times more successful than agents not using shout-ahead. By comparing the performance of shout-ahead and shout-ahead-less agents in a different context, a traffic network for instance, would lend more insight on shout-ahead’s general effectiveness as an agent architecture. Should it improve agent performance by a similarly significant margin, we can begin to confidently hypothesize that learning cooperative behavior with shout-ahead holds merit in a wide array of applications, for at the moment it only has one proven application space.

Before experimental evaluation can take place, the agent pools must learn. Given the dynamic nature of traffic networks, individuals (or, solutions to the problem) will learn on the same networks they are tested on, with varying traffic flows adjusted in various manners (as will be discussed later in this section). The exact number of networks to be used for learning and experimental evaluation is not set at this time, but they quantity will likely range between 3 and 5. The complexities of road networks in SUMO can vary widely, just as they do in the real world. For the purpose of this project, the most simplistic road network 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.

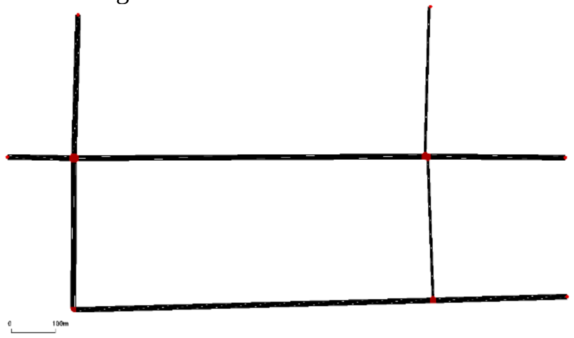

Fig. 7. Example of a simple road network in SUMO.

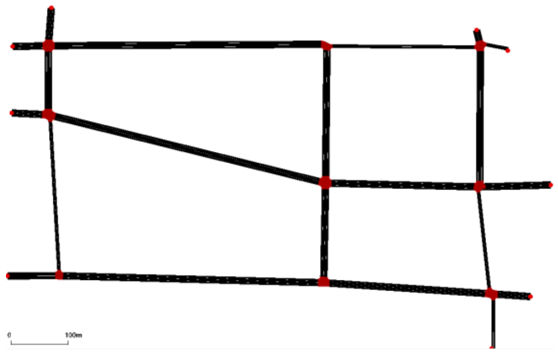

Fig. 8. Example of a complex road network in SUMO.

In SUMO, intersections are represented by 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. The maximum complexity of a SUMO road network for the purposes of this project is a difficult thing to define, given there a number of constraints to consider when doing so. Firstly, we must consider the hardware constraints. The number of intersections of a road network has a direct relationship with the maximum number of vehicles that can be going through the network at any one time, meaning computational complexity increases significantly with the introduction of every intersection. The second constraint to consider is time; the more complex a road network is, the longer it will take for the system to complete training and testing runs. Given the relatively limited timeframe this project has to be completed in (approximately 3 months from the time of this writing), we must be wary when designing a road network For the time being, we have decided the most complex road network to be used will feature no more than three of each of our chosen intersection types. An example of such a network is in Fig. 8.

Finally, we must consider what traffic flows will be used for both learning and testing. This is not a trivial matter, and will require some thought, and trial and error to decide upon. Training will require multiple different traffic flows, however the differences between these must be carefully considered as well. Emulating volume at different times of day and the traffic that comes with it would be ideal, however using starkly different flows for the training of one agent pool, such as morning and afternoon rush hour, may just confuse the learner and be ultimately counterproductive. It is likely that the differences between flows will be minimal for learning, and the same ones will be utilized in testing. Additionally, new flows will designed for testing, their variance from the learning ones remaining in question as well. We simply do not have enough concrete information about what constitutes a useful flow at this time to make any concrete assertions.

Once we have decided on our networks for learning and testing, and the accompanying traffic flows, we will evaluate the performances of the three types of traffic signal control agents (timing-cycle, shout-ahead enabled and shout-ahead disabled) to see which proves superior in which circumstances. As mentioned previously, the performance of an agent-type will be measured by how quickly they can facilitate a certain flow through the network. That is, the time it takes to get every vehicle in the network from its starting point to its destination. As the experimental evaluation goes on, there will likely be results that motivate new experiments to be carried out and adjustments to be made to our shout-ahead implementation.

V. Related Work

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 [11] showed that using an intelligent agent to generate traffic signal controller (TSC) policies was more effective. Similar findings were presented in [12], where agents learned a near-optimal policy. Furthermore, [15] 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 [9], [10], [12], [13], [17], [18], [19], [20], [22], [23] and [24]. Naturally, nuances exist between each of the studies, despite their shared solution to the TSC problem. [9] 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. [10] and [19] make use of similar neighbor communication approaches, except [10] 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 [9] and [19], [10] applies its solution directly to a large-scale problem to verify its efficacy in realistic settings, and found success with this approach.

Cooperation between individual TSC agents wasn't the only way MARL was applied to the TSC problem. [14] defined two individual agent classes which worked together to optimize traffic flow. They presented a detailed protocol making use of two different classes of agents, dubbed intersection managers and driver agents. The presence of the latter agent marked a departure from the singular focus applied on the TSC agent by most other papers. [23] meanwhile assigned multiple local controller agents to a single traffic junction, making use of local traffic state data, a learned probabilistic model of car behavior, and a learned value function which indicates how traffic light decisions affect long-term utility, in terms of the average waiting time of cars, to optimize the performance of the intersection.

Some algorithms expanded upon the basic MARL strategies implemented by papers mentioned above. [18] for instance, detailed a Q-learning algorithm with a feedforward neural network for value function approximation. This approach was shown to be a superior method in this context when compared against a longest-queue first algorithm. Examples of other solutions that make a point of their decentralized nature are [15], [16], [21], and [25]. [25] details a fully scalable and decentralized MARL algorithm for an advantage actor critic agent within the context of adaptive traffic signal control, as a proposed evolution to past solutions to the TSC problem using MARL. [21] meanwhile proposed a TSC system enabled by a hierarchical multi-agent modeling framework. In a similar vein, [15] proposed a new method called Reinforcement Learning with Context Detection (RL-CD), which it claimed would be superior to popular methods such as Q-learning due to its adaptability in dynamic environments, such as intersections. RL-CD employs multiple partial models of the environment to allow the learning system to partition its knowledge in a way that each model becomes responsible for understanding one kind of flow pattern. Empirical results supported the authors claim.

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. [24] 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

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, 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. Theoretically, the ability for a traffic signal control agent to learn which actions are proper in which situations, coupled with communication with its neighboring cohorts, should result in more optimal actions being taken, and more effective intersection control than a near-static algorithm would provide.

References

[1] M. Bennardo, "StatsCan study shows Canadian commute times are getting longer — and it's costing us," *CBC*, March 2019. [Online]. Available: https://www.cbc.ca/news/business/statistics-canada-commute-times-study-1.5038796. [Accessed: Sept. 21, 2019].

[2] T. Yaropud, J. Gilmore, S. LaRochelle-Côté, "Results from the 2016 Census: Long commutes to work by car," *Statistics Canada*, February 2019. [Online]. Availble: https://www150.statcan.gc.ca/n1/en/pub/75-006-x/2019001/article/00002-eng.pdf?st=wRNvAal2. [Accessed: Sept. 21, 2019].

[3] Y. Shaoxin, X. Zhao, 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] National Academies of Sciences, Engineering, and Medicine, Signal Timing Manual – Second Edition, Washington, DC: The National Academies Press, 2015.

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

[6] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "SUMO– simulation of urban mobility: an overview," in Proceedings of the Third International Conference on Advances in System Simulation (SIMUL 2011), 2011.

[7] "Automatically Generated TLS-Programs," Simulation/Traffic Lights – SUMO Documentation. [Online]. Available: https://sumo.dlr.de/docs/Simulation/Traffic_Lights.html#actuated_traffic_lights. [Accessed: 17-Jan-2020].

[8] R.S. Sutton and A.G. Barto, Reinforcement Learning: An Introduction, The MIT Press, 1998.

[9] K. J. Prabuchandran, A. N. Hemanth Kumar and S. Bhatnagar, "Multi-agent reinforcement learning for traffic signal control," *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Qingdao, 2014, pp. 2529-2534.

[10] L. Kuyer, S. Whiteson, B. Bakker, and N. Vlassis, "Multiagent Reinforcement Learning for Urban Traffic Control Using Coordination Graphs," *Machine Learning and Knowledge Discovery in Databases Lecture Notes in Computer Science*, pp. 656–671, 2008.

[11] M. Abdoos, N. Mozayani and A. L. C. Bazzan, "Traffic light control in non-stationary environments based on multi agent Q-learning," *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Washington, DC, 2011, pp. 1580-1585.

[12] P. Gregoire, C. Desjardins, J. Laumonier and B. Chaib-draa, "Urban Traffic Control Based on Learning Agents," *2007 IEEE Intelligent Transportation Systems Conference*, Seattle, WA, 2007, pp. 916-921.

[13] M. Wiering, "Multi-Agent Reinforcement Learning for Traffic Light Control," *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, Stanford, CA, 2000.

[14] K. Dresner and P. Stone, "Multiagent Traffic Management: Opportunities for Multiagent Learning," *Learning and Adaption in Multi-Agent Systems Lecture Notes in Computer Science*, pp. 129–138, 2006.

[15] D. de Oliveira Boschetti, A. Bazzan, B. da Silva, E. Basso and L. Nunes, "Reinforcement Learning based Control of Traffic Lights in Non-stationary Environments: A Case Study in a Microscopic Simulator," *Proceedings of the 4th European Workshop on Multi-Agent Systems EUMAS'06*, Lisbon, Portugal, 2006.

[16] E. Camponogara and W. Kraus, "Distributed Learning Agents in Urban Traffic Control," *Progress in Artificial Intelligence Lecture Notes in Computer Science*, pp. 324–335, 2003.

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

[18] I. Arel, C. Liu, T. Urbanik and A. G. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control," *IET Intelligent Transport Systems*, vol. 4, no. 2, pp. 128-135, 2010.

[19] E. Van der Pol and F.A. Oliehoek, "Coordinated Deep Reinforcement Learners for Traffic Light Control," *NIPS'16 Workshop on Learning, Inference and Control of Multi-Agent Systems*, December 2016.

[20] D. Houli, L. Zhiheng, and Z. Yi, "Multiobjective Reinforcement Learning for Traffic Signal Control Using Vehicular Ad Hoc Network," *EURASIP Journal on Advances in Signal Processing*, vol. 2010, no. 1, 2010.

[21] J. Jin and X. Ma, "Hierarchical multi-agent control of traffic lights based on collective learning," *Engineering Applications of Artificial Intelligence*, vol. 68, pp. 236–248, 2018.

[22] S. El-Tantawy and B. Abdulhai, "Multi-Agent Reinforcement Learning for Integrated Network of Adaptive Traffic Signal Controllers (MARLIN-ATSC)," *2012 15th International IEEE Conference on Intelligent Transportation Systems*, 2012.

[23] B. Bakker, S. Whiteson, L. Kester, and F. C. A. Groen, "Traffic Light Control by Multiagent Reinforcement Learning Systems," *Interactive Collaborative Information Systems Studies in Computational Intelligence*, pp. 475–510, 2010.

[24] D. D. Oliveira and A. L. Bazzan, "Multiagent Learning on Traffic Lights Control: Effects of Using Shared Information," *Multi-Agent Systems for Traffic and Transportation*, pp. 307–321, 2009.

[25] T. Chu, J. Wang, L. Codeca and Z. Li, "Multi-Agent Deep Reinforcement learning for large-scale Traffic Signal Control," *IEEE Transactions on Intelligent Transportation Systems*, 2019.