IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Dynamic Graph Configuration for Self-Driving Cars

Ashish Chaurasiya, Devesh Khade, Kapil Belure, Anuj Mishra,
Prof. M. H Ranadive
Department of Computer Engineering
D.Y Patil Institute of Engineering and
Technology, Ambi, Pune
Savitribai Phule Pune University

Abstract— Conventional traffic optimization solutions expect that the graph structure of street systems is static and missing for further traffic flow optimization. We are interested in optimizing traffic flows as a latest type of graph-based issue, where the graph structure of a street network can adjust to traffic conditions in real time. Specifically, we focus on the dynamic setup of traffic-lane directions, which can offer assistance in adjusting the utilization of traffic paths in opposite directions. The rise of connected autonomous vehicles offers an opportunity to apply this kind of dynamic traffic optimization on a huge scale. The existing procedures for optimizing lane directions are however not appropriate for dynamic traffic situations due to their high computational complexity and the static nature.

In this paper, we propose an effective traffic optimization solution, called Coordinated Learning-based Lane Allocation (CLLA), which is appropriate for dynamic setup of path directions. CLLA comprises of a two-layer multi-agent architecture, where the bottom-layer agents use a machine learning technique to discover a appropriate arrangement of lane-directions around street crossing points. The lane-direction changes proposed by the learning agents are at that point coordinated at the next level to diminish the negative impact of the changes on other parts of the street network. Our test results show that CLLA can decrease the average travel time significantly in congested street networks. We accept our method is common enough to be applied on other types of networks as well.

Keywords—Graphs, CLLA, Reinforcement Learning

INTRODUCTION

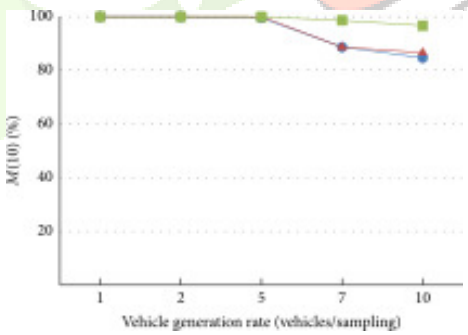
The objective of traffic optimization is to improve traffic flows in street networks. Conventional solutions ordinarily assume that the structure of street networks is static regardless of how traffic can change at real time. A less-common way to optimize traffic is by performing limited changes to street networks which when is-use are sent in very little scale. We focus on dynamic lane-direction changes, which can offer assistance in adjusting the utilization of traffic paths in numerous circumstances, e.g., as soon as when the traffic paths in one direction become congested whereas the traffic paths in the opposite direction are underused. Unfortunately, the existing procedures for optimizing path- directions are not appropriate for dynamic traffic environment on huge scale due to their high computational complexity. We create an effective solution for optimizing lane-directions in highly dynamic traffic situations. Our solution is based on an algorithm that alters the property of a street network graph for improving traffic flow within the corresponding street network, introducing a new graph issue.

(a)Traffic before path- direction change

(b)Traffic after path- direction change

The impact of dynamic lane-direction configurations can be shown in the following example, where vehicles are moving north-bound and 1 vehicle are moving south-bound (Figure 1) at a certain time. In Figure 1a, there are 4 north-bound lanes and 4 south-bound lanes. Due to the large number of vehicles and the limited number of lanes, the north-bound traffic is highly congested. At the same time, the south-bound vehicles are moving at a high speed as the south-bound lanes are almost empty. Figure 1b shows the dramatic change of traffic flow after lane-direction changes are applied when congestion is observed, where the direction of E, F and G is reversed. The north-bound vehicles are distributed into the additional lanes, resulting in a higher average speed of the vehicles. At the same time, the number of south-bound lanes is reduced to 1. Due to the low number of south-bound vehicles, the average speed of south-bound traffic is not affected. The lane-direction change helps improve the overall traffic efficiency in this case. This observation was used by traffic engineers of certain street sections for many years and applied in a more static way. We aim to scale this to extreme levels in time and space. The advantage of dynamic lane-direction changes can also be observed in preliminary tests, where we compare the average travel time of vehicles in two scenarios, one uses dynamic lane-direction configurations, another uses inactive lane direction configurations. The dynamic lane-direction configurations are computed with a direct solution. The result shows that lane-direction changes diminish travel times

Figure 2: The average travel time of vehicles when using static and dynamic path-direction configurations.



by 14% on average when the traffic increases (see Figure 2). In traffic engineering terms usually, there is a dramatic reduction. In spite of their potential advantage, dynamic lane direction changes cannot be viably applied to existing traffic systems as they require additional signage and security features. However, the rise of connected autonomous vehicles (CAVs) can make dynamic lane-direction changes a common practice in the future. Our past work shows that CAVs have the potential to enable inventive traffic management solutions. Compared to human-driven vehicles, CAVs are more capable of reacting to a given command in a timely manner. CAVs can moreover give point by point traffic telemetry data to a central traffic management system in real time. This helps the system to adjust in dynamic traffic conditions.

We formulate path allocation based on real-time traffic as a modern graph issue with the aim to discover a new graph from a street network (i.e., dynamically optimize the graph) such that total travel time of all vehicles in the street network is minimized. In order to optimize the flow of the entire network, all the traffic paths within the network must be considered. In numerous circumstances, one cannot simply allocate more traffic lanes at a road segment for a particular direction when there's more traffic request in the direction. This is because a lane-direction change at a street section can affect not only the flow in both directions at the street segment but also the flow at other street sections. Due to the complexity of the issue, the computation time can be very high with

the existing approaches as they aim to discover the optimal configurations based on linear programming, and thus are not appropriate for frequent precomputation over huge networks.

To address the above-mentioned issues, we propose a light-weight and efficient system, called a Coordinated Learning-based Lane Allocation (CLLA) framework, for optimizing lane-directions in dynamic traffic situations. The CLLA approach finds the configurations that efficiently improve the traffic efficiency of the entire network, while keeping the computation cost of the solution low. The key point is that traffic optimization can be decoupled into two forms: i) a local process that proposes lane-direction changes based on local traffic conditions around street intersections, and b) a global process that evaluates the proposed lane-direction changes based on their large-scale impact.

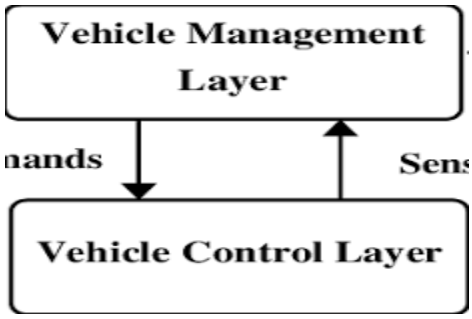


Figure 3: The hierarchical architecture of our traffic management solution based on path-direction changes.

The architecture of our hierarchical solution is outlined in Figure 3. The bottom layer comprises of a set of autonomous agents that operate at the intersection level. An agent finds appropriate lane-direction changes for the street sections that connect to a particular intersection. The agent uses reinforcement learning, which helps decide the best changes based on numerous dynamic components. The agents send the proposed lane direction changes to the upper layer, which comprises of a coordinating agent. The coordinating agent maintains a data structure, named Path Dependency Graph (PDG), which is built based on the trip information of connected autonomous vehicles. With the help of the data structure, the coordinating agent evaluates the global impact of the proposed path direction changes and chooses what changes ought to be made to the traffic lanes. The decision is sent back to the bottom layer agents, which will make the changes accordingly. The main contributions of our work are as follows:

- We formulate dynamic path allocation as a new graph issue (Dynamic Resource Allocation problem).
- We propose a hierarchical multi-agent solution (called CLLA) for efficient dynamic optimization of path- directions that uses reinforcement learning to capture dynamic changes in the traffic.
- We introduce an algorithm and innovative data structure (called path dependency graph) for coordinating path- direction changes at the global level.
- Extensive experimental assessments shows that CLLA significantly beats other traffic management solutions, making it a practical tool for moderating traffic congestion for future traffic networks.

RELATED WORK

A. Traffic Optimization Algorithms Existing traffic optimization algorithms are commonly based on traffic flow optimization with linear programming. The algorithms are appropriate for the circumstances where traffic request and congestion levels are moderately static. When there's a significant change within the network, the optimal solutions normally ought to be re-computed from scratch. Due to the high computational complexity of finding an optimal solution, the algorithms may not be appropriate for highly dynamic traffic environments. With the rise of reinforcement learning, a new generation of traffic optimization algorithms have risen. In reinforcement learning, a learning agent can discover the rules to achieve an objective by repeatedly interacting with an environment. The interactive process can be displayed as a finite Markov Choice Prepare, which needs a set of states S and a set of actions A per state. Given a state s of the environment, the agent takes an action a. As the result of the action, the environment state may change to sJ with a reward r. The agent then decides the next action in order to maximize the reward in the next round. Reinforcement learning-based approaches can recommend the best actions for traffic optimization given a combination of network states, such as the queue size at intersections. They have an advantage over linear programming-based approaches, since in case trained well, they can optimize traffic in a highly dynamic network. In other words, there's no need to re-train the agent when there's a change in the network. Different to the existing approaches, our solution uses reinforcement learning for optimizing lane-directions. A common issue with reinforcement learning is that the state space can grow exponentially when the dimensionality of the state space grows linearly. For example, let us assume that the initial state space has only one dimension, the queue size at intersections. In case we add two dimensions to the state space, traffic signal stage and traffic path configuration, there will be three dimensions and the state space is four times as large as the original state space. The fast growth of the state space can make reinforcement learning unacceptable for real deployments. This issue is known as the curse of dimensionality. A common way to

mitigate the issue is by using a function approximator such as a neural organize. Such methods have been mainly used for dynamic traffic signal control, while we expand the use of the method to dynamic path- direction configuration.

B. Numerous existing traffic optimization solutions use model- based reinforcement learning, where one must know the exact probability that a particular state travels to another particular state as a result of a particular action. Nonetheless, such an assumption is unrealistic since the complete knowledge of state transition probabilities can barely be known for highly complex traffic systems. Different to model-based approaches, our optimization solution utilizes a model-free algorithm, Q- learning, which does not require such knowledge and thus is much more appropriate to real traffic systems. Coordination of multi-agent reinforcement learning can be accomplished through a joint state space or through a coordination graph. Such procedures however require agents to be trained on the targeted network. Since our approach uses an implicit mechanism to coordinate, once an agent is trained, it can be used in any street network. *Lane-direction Configurations*

Research shows that dynamic lane-direction changes can be an effective way to improve traffic efficiency [3]. However, existing approaches for optimizing lane-directions are based on linear programming [5]–[7], [27], which are unsuitable for dynamic traffic situations contributes to their high computational complexity. Their experiments show that the total travel time can be decreased. In any case, the computational time grows exponentially when the number of vehicles grows linearly, which can make the approach unsuitable for highly dynamic traffic situations. Other approaches perform optimization based on two processes that interact with each other. One process is for minimizing the entire system cost by reversing path directions while the other process is for making route decisions for individual vehicles such that all the vehicles can minimize their travel times. To discover a great optimization solution, the two processes ought to be interact with each other iteratively. The high computational cost of the approaches can make them unsuitable for dynamic traffic optimizations. Moreover, all these approaches assume exact knowledge of traffic request over the time horizon is known previously; this assumption does not hold when traffic request is stochastic. On the contrary, CLLA is lightweight and can adjust to highly dynamic circumstances based on reinforcement learning. The learning agents can discover the viable lane-direction changes for individual street intersections even when traffic request changes dramatically.

C. Traffic Management with Connected Autonomous Vehicles

Some recent development of traffic management solutions is custom fitted for the period of connected autonomous vehicles. We have created a traffic management model that combines connected autonomous vehicles and intelligent street framework for improving traffic flow . We have created an approach to improve traffic efficiency at intersection using connected autonomous vehicles. We use the CAVs as an opportunity for path optimization.

PROBLEM DEFINITION

In this section, we formalize the issue of traffic optimization based on dynamic configuration of path directions. Our issue is similar to Network Design Problem, however NDP is based on the assumption of knowledge of traffic request for entire time horizon at time zero and the output network is designed for a common state. We attempt to configure a graph (street network) at regular time intervals based on real-time traffic, hence we name this issue, Dynamic Graph Resource Allocation problem. Let tt(V, E) be a street network graph, where V is a set of vertices and E is a set of edges. Let us assume that edge e E connects between vertex x V and vertex y V . The edge has three properties. The first property is that the total number of paths, ne, which is a constant number. The second property is the number of paths that begin from x and end in y, ne1 . The third property is the number of paths in the opposite direction (from y to x), ne2 . ne1 and ne2 can change but they are always subject to the following constraint. ne1 + ne2 = ne (1) We assume that a CAV follows a pre-determined path based on an origin-destination (O-D) pair. Let the number of unique O-D pairs of the existing vehicles be k at a given time t. For the ith (i <= k) O-D pair, let di,t be the traffic request at time t, i.e., the number of vehicles with the same O-D pair at that time. The traffic request can be stochastic. Let the travel time of vehicle j with the ith O-D pair be TTi,j, which is the duration for the vehicle to move from the origin to destination. For a given time t, the average travel time of all the vehicles, which will reach their goal during a time period T after t, can be defined as

$$ATT_{<t,t+T>} = \sum_{i=1}^{\Sigma} \sum_{j=1}^{\Sigma} TT_{i,j} / \sum_{i=1}^{\Sigma} m_i$$

where m_i is the number of vehicles with the i^{th} O-D pair that will complete their trips between t and $t + T$.

We define and solve a form of the issue where at frequent regular intervals, we optimize travel time, while changing the path arrangement in all edges. We discover a new graph $tt^{'}_{t}(V, E^{'})$ at a given time t from previous tt at previous time step. Let $e^{'}_{1}$, $e^{'}_{2}$ $E^{'}$ and $e^{'}_{1}$ connects from vertex x to vertex y and $e^{'}_{2}$ connects vertex y to vertex x. We discover for all edges the values of $n_e t$ and $n_e t$, such that the $\in$ average travel time $ATT_{<t,t+T>}$ is minimized. We call this Dynamic Resource Allocation problem.

1 2

DEMAND-BASED LANE ALLOCATION (DLA)

When considering dynamic lane-direction changes, a straightforward solution can use a centralized approach to optimize lane-directions based on the full knowledge of traffic demand, i.e., the number of vehicle paths that pass through the street links. We call this solution *Demand-based Lane Allocation (DLA)*. Algorithm 1 shows the implementation (in pseudo code) of such idea to compute the configuration of lane-directions. DLA allocates more paths for a particular direction when the average traffic request per path in the direction is higher than the average traffic request per path in the opposite direction. To specify the directions, we define two terms, *upstream* and *downstream*. The terms are defined as follows. Let us assume that all the vertices of the street network graph are ordered by the identification number of the vertices. Given two vertices, v_1 and v_2, and a direction that points from v_1 to v_2, we say that the direction is **upstream** if v_1 is lower than v_2 or **downstream** if v_1 is higher than v_2.

DLA first computes the traffic demand at the edges that are on the path of the vehicles. The traffic request is computed for the upstream direction (up_e) and the downstream direction ($down_e$) separately. Then it evaluates the difference of the average traffic request per path between the two directions (Line 9-11). Based on the difference between the two directions, DLA decides whether the number of paths in a specific direction should be increased. We should note that increasing the number of paths in one direction suggests that the number of paths in the opposite direction is decreased. DLA only decreases the number of paths in a direction in case the traffic request in that direction is lower than a threshold. The complexity of the algorithm is (k E), where E is the number of edges in tt and k is the number of O-D pairs. While it is clear to implement and is efficient, there are two notable disadvantages of DLA. First, the algorithm does not consider real-time traffic conditions, such as the queue length at a given time, during optimization; the only data used for optimization is (assumed apriori known) traffic request and exact knowledge of traffic request is troublesome to obtain in dynamic street networks. This can make the lane-direction configuration less adaptive (and less applicable) to real-time traffic conditions. Second, the lane-direction optimization for individual road segments is performed individually, not considering the potential impact of a lane-direction change at a road section on other street sections in the same street network. Hence, a lane-direction change that helps improve traffic efficiency at a street link may lead to the decrease of traffic efficiency in other parts of the street network.

COORDINATED LEARNING-BASED LANE ALLOCATION (CLLA)

To handle the issues of the direct solution, we propose a fundamentally different solution, a Coordinated Learning-based Lane Allocation (CLLA) framework. CLLA uses a machine learning procedure to assist optimize path- direction configurations, which permits the framework to adjust to a high variety of real-time traffic conditions. In addition, CLLA coordinates the lane-direction changes by considering the effect of a potential lane-direction change on different parts of the street network. DLA, on the other hand, does not consider the global impact of lane-direction changes. Another difference between the two is that DLA requires the full path of vehicles to be known for computing traffic request. CLLA only needs to know partial data almost vehicle paths in addition to certain data around real-time traffic conditions, such as intersection queue lengths and path configuration street sections which can be obtained from inductive-loop traffic detectors. CLLA uses a two-layer multi-agent architecture. The bottom layer comprises of learning agents that are responsible for optimizing the direction of paths connected to particular intersections. Using the multi-agent approach can significantly boost the speed of learning. The lane-direction changes that are chosen by the learning agents are aggregated and evaluated by a planning agent at the upper layer, which will send the globally optimized lane-direction configuration to the bottom layer for making the changes. A more detailed overview of CLLA is shown in Figure 4. As the figure shows, an agent in the bottom layer only observes the nearby traffic condition around a particular intersection. Agents make decisions on lane-direction changes independently.

Algorithm 1: Demand-based Lane Allocation (DLA)

Input: A road network graph $tt(V, E)$.
Input: The set of paths. A path is a sequence of edges on the shortest path between a specific Origin-Destination (O-D) pair. The set of paths includes the paths of all unique O-D pairs.
Input: The demands associated with the paths, where a demand is the number of vehicles that follow a specific path.
Input: th: demand threshold.
Input: g: minimal gap in traffic demand that can trigger a lane-direction change.

```
1  foreach p ∈ paths do
2      foreach e ∈ p do
3          if p passes e in the upstream direction then
4              up_e += demand of p
5          if p passes e in the downstream direction then
6              down_e += demand of p
7  foreach e ∈ E do
8      minLoad ← min(up_e, down_e)
9      down'_e ← down_e / number of downstream lanes
10     up'_e ← up_e / number of upstream lanes
11     gap ← (down'_e − up'_e) / (up'_e + down'_e)
12     if minLoad < th then
13         if gap > g then
14             move one upstream lane to the set of
               downstream lanes
15         if gap < −g then
16             move one downstream lane to the set of
               upstream lanes
```

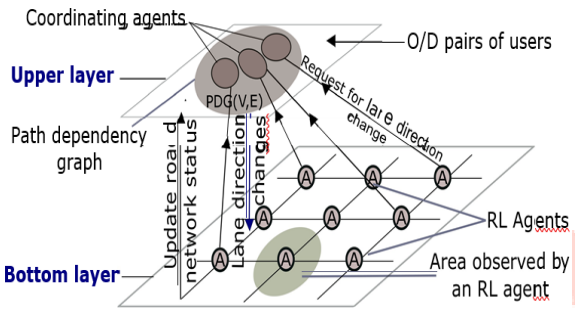


Figure 4: An iutline of the CLLA's architecture

Whenever an agent makes a path- direction change, it sends the proposed change to the coordinating agent in the upper layer. The agents moreover send certain traffic data to the upper layer periodically. The data can offer assistance indicating whether there's an imbalance between upstream traffic and downstream traffic at particular street sections. The coordinating agent evaluates whether a change would be beneficial at the global level. The evaluation process includes a novel data structure, Path Dependency Graph (PDG), to inform decisions sent from the foot layer. The coordinator may permit or deny a lane-direction change request from the bottom-layer. It may moreover decide to make further changes to lane-directions in addition to the requested changes. After evaluation, the coordinating agent informs the bottom-layer agents about the changes to be made.

We should note that the coordinator doesn't need to evaluate a lane-direction change request as soon as it arrives. As shown in Figure 5, the coordinator evaluates the path direction changes periodically. The time interval between the evaluations is T . All the requests from the bottom-layer agents are buffered during the interval. The exact value of the interval must be balanced case by case. A short interval may increase the computational cost of the solution. A long interval may decrease the viability of the optimization.

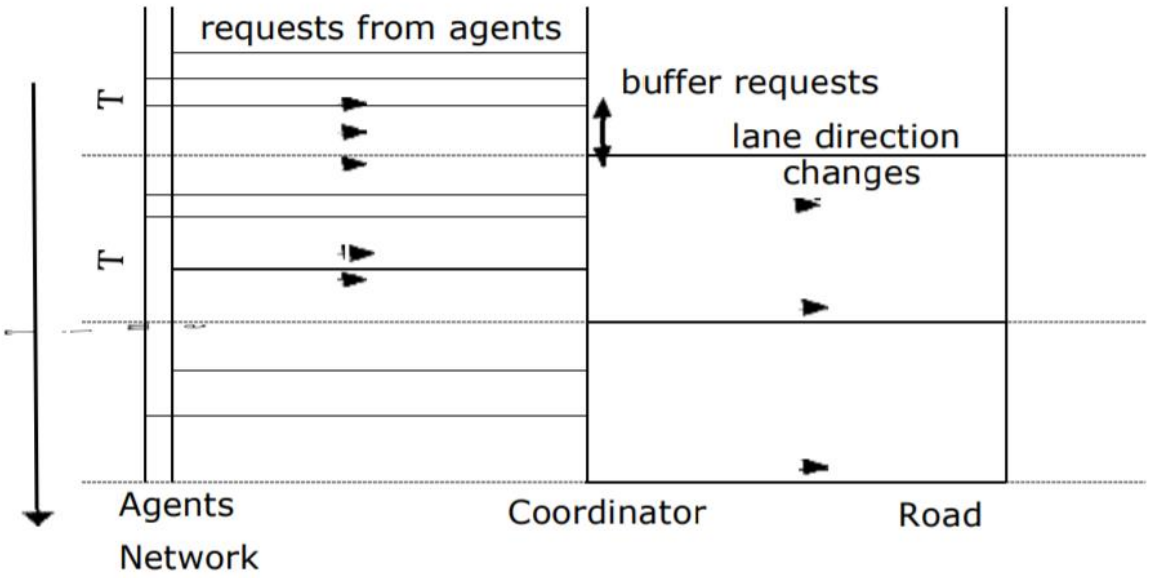


Figure 5: The CLLA's inter-communication timeline between

A. CLLA Algorithm

Algorithm 2 shows the entire optimization process of CLLA. During one iteration of the algorithm, each learning agent finds the lane-direction changes around a specific street intersection using the process described in Section V-B. The proposed change is stored as an edge-change pair, which is buffered in the system. When it is time to evaluate the proposed changes, the system uses the Direction-Change Evaluation algorithm (Section V-C) to quantify the conflicts Evaluation algorithm (Section V-C) to quantify the conflicts between the proposed changes. For example, when a learning agent proposes to increase the number of upstream paths on street section s1 whereas another agent proposes a lane direction change on a different street section s2, which can lead to the increase of the downstream traffic flow on s1, there is a conflict between the proposed changes for s1. The Change Evaluation algorithm also expands the set of the proposed changes that may be beneficial. Upon returning from the Change Evaluation algorithm, CLLA checks the expanded set of edge-change pairs. For each edge-change pair, if the number of conflicts for the edge are below a given limit, the change is applied to the edge (Line 12).

B. Learning-based Lane-direction Configuration

In the CLLA framework, the bottom-layer agents use the Qlearning technique to find appropriate lane-direction changes based on real-time traffic conditions. Q-learning aims to discover a approach that maps a state to an action. The algorithm depends on an action value function, Q(s, a), which computes the quality of a state-action combination. In Qlearning, an agent tries to discover the optimal approach that leads to the maximum action value. Q-learning updates the action value function using an iterative process as shown in Equation 3.

Algorithm 2: Coordinated Lane Allocation (CLLA)

Input: T, action evaluation interval
Input: $EC_{initial}$, set of edge-change pairs proposed by the learning agents
Input: $EC_{expanded}$, set of edge-change pairs given by the coordinator
Input: mc, the maximum number of conflicts between lane-direction changes before a proposed lane-direction change can be applied to an edge.
Input: $tt(V, E)$, a road network graph.
Input: l, the lookup distance for building Path Dependency Graph.
Input: dp, the maximum search depth in Path Dependency Graph for evaluating lane-direction changes.

```
while True do
    foreach agent ∈ Agents do
        determine the best lane-direction change for all
          the edges (road segments) that connect to the
          vertex (road intersection) controlled by the
          agent
        foreach edge e that needs a lane-direction
          change do
            EC_initial.insert({e, change})
    if T = t then
        t ← 0
        EC_expanded ← Direction-Change Evaluation
          (EC_initial, tt, l, dp)
        EC_initial ← ∅
        foreach {e, change} ∈ EC_expanded do
            if number of conflicts on e is less than mc
              then
                apply the lane-direction change to e
    t ← t + 1
```

$Q^{new}(s,a)=(1-\alpha)Q_t(s,a)+\alpha(r_{t+1}+\max Q(s_{t+1},a))$

where s is the current state, a is a specific action, st+1 is the next state as a result of the action, maxQ(st+1, a) is the a estimated optimal action value in the next state, value rt+1 is an observed reward at the next state, α is a learning rate and γ is a discount factor. In CLLA, the states, actions and rewards used by the learning agents are defined as follows.

1) States: A learning agent can work with four types of states. The first state represents the current traffic signal stage at an intersection. The second state represents the queue length of incoming vehicles that are going to pass the intersection without turning. The third state represents the queue length of incoming vehicles which are going to turn at the intersection. The fourth state represents the queue length of outgoing vehicles, i.e., the vehicles that have passed the intersection.

Although it is possible to add other types of states, we discover that the combination of the four states can work well for traffic optimization.

2) Actions: There are three possible actions: increasing the number of upstream paths by 1, increasing the number of downstream paths by 1 or keeping the current configuration. When the number of paths in one direction is increased, the number of paths in the opposite direction is decreased at the same time. Since a learning agent controls a particular street intersection, the agent determines the action for each individual street section that connects with the crossing point. An agent is permitted to take a lane-changing action only when there's a traffic imbalance on the street section (see Equation 4 for the definition of traffic imbalance).

3) Rewards: We define the rewards based on two components. The first component is the waiting time of vehicles at an intersection. When the waiting time decreases, there's generally an improvement of traffic efficiency. Thus, the rewards should consider the difference between the current waiting time and the updated waiting time of all the vehicles that are approaching the intersection. The second component is the difference between the length of vehicle queues at different approaches to an intersection. When the queue length of one approaching street is significantly longer than the queue length of another approaching street, there's a higher chance that the activity becomes congested in the previous case. Hence, we have to be penalized the actions that increase the difference between the longest queue length and the shortest line length. The following reward function combines the two variables. A parameter β is used to provide weights for the two components. We normalized the two components to stabilize the learning.

$$R = (1-\beta) \times \frac{\text{Current wait time} - \text{Next wait time}}{\max(\text{Next wait time}, \text{Current wait time})} - \beta \times \frac{\text{Queue length difference}}{\text{Aggregated road capacity}}$$

C. Coordination of Lane-direction Changes

We create the coordinating process based on the observation that a locally optimized lane-direction change may conflict with the lane-direction changes that happen in the surrounding regions. A conflict can happen due to the fact that the impact of a lane-direction change can spread from one street section to other street section. For example, let us assume that a constant section of the upstream traffic that passes through street section x will also pass-through street section y in the upstream direction afterward. An increase of the upstream paths on x can lead to a significant increase of upstream traffic on x due to the increased traffic capacity in the direction. Over time, the traffic volume change on x can lead to the increase of the upstream traffic on y, which implies that the number of upstream paths at y may need to be increased to suit the change of traffic volume. In this case, the lane direction change at y can be seen as a considerable change caused by the change at x. However, the learning agent that controls the lane directions at y may recommend an increase of downstream paths based on the current nearby traffic condition at y. If this is the case, the locally optimized change will conflict with the considerable change. The key task of the coordinating process is evaluating such conflicts in street networks. If there are a large number of conflicts at a street sections, the locally optimized change should not be applied since it may have a negative impact on the traffic flows at the global level later on. This is a key point behind the coordination process of our solution. As shown in Section V-A, our solution applies a proposed lane-direction change to a street

section only when the number of the conflicts is below a given threshold.

To help identify the conflicts between lane-direction changes, we create a novel data structure, named *Path Dependency Graph (PDG).* The data structure maintains various types of traffic data, including the path of traffic flow, the proposed lane-direction changes and the current traffic conditions. The coordinating agent uses PDG to search for the locations of consequential lane-direction changes. The conflicts between lane-direction changes are then identified by comparing the consequential lane-direction changes and the proposed lane-direction changes at the same locations. The coordinating agent also proposes additional lane-direction changes using PDG.

A PDG (PDtt(V P DG, EP DG)) comprises of a number of vertices and a number of directional edges. A vertex v V P DG represents a street section. The corresponding street sections of the two vertices must appear in the way of a vehicle. A vertex can connect to a number of outdegree edges and a number of in-degree edges. The direction of an edge depends on the order of traffic stream that goes through the two street sections. An edge that begins from vertex v1 and ends in vertex v2 shows that the traffic flow will pass through v1's corresponding street section first then pass through v2's corresponding street section afterward. We should also note that the two street sections, which are linked by an edge, don't need to share a common street intersection, i.e., they can be disjoint. Given the path of all the vehicles, a PDG can be built such that all the unique street sections on the vehicle paths have corresponding vertices in the graph. For each pair of the street sections on a vehicle path, there is a corresponding edge in the graph. If the path of two or more vehicles goes through the same pair of street se, there is only one corresponding edge in the graph.
A vertex of PDG has the following properties.

- **Proposed Change**: The proposed lane-direction change at the corresponding road segment. This may be submitted from a learning agent or created by the system during the coordinating process. The property value can be 1, 0 and −1. A value of 1 means the upstream direction gets one more lane. A value of 0 means there is no need for a change. A value of −1 means the downstream direction gets one more lane. **Consequential Changes:** A list of potential lane direction changes caused by lane-direction changes at other street sections. Similar to the Proposed Change property, the value of a considerable change can be 1, 0 and −1.
  Imbalance: The path direction which has a considerably higher traffic load than the other direction. The property value can be upstream, downstream and none. In our implementation, the imbalance of traffic load is measured based on the queue length in the opposite directions. Let q_{up} be the upstream queue length and q_{down} be the downstream queue length. Let the total queue length in both directions be q_{total}. Let P be a threshold percentage. The property value is computed as follows.

$$\text{Imbalance} = \begin{cases} \text{upstream}, & \text{if } q_{up}/q_{total} > P \\ \text{downstream}, & \text{if } q_{down}/q_{total} > P \\ 0, & \text{otherwise} \end{cases}$$

Due to the dynamic nature of traffic, imbalance value may change frequently, leading to frequent changes of lane directions. This may not be ideal in practice. One can get a steady imbalance value by adding certain restrictions in the computation. For example, one may require that the ratio between upstream queue length and the total queue length must be above the threshold for a certain period of time before setting the imbalance value to upstream.
Current Lane Configuration: The number of upstream lanes and the number of downstream lanes in the corresponding road segment.

An edge of PDG incorporates a property called impact, which shows whether a lane-direction change at the beginning vertex can lead to the same change at
the ending vertex. The value of this property can be -1 or 1. A value of 1 implies the change at both vertices will be the same. For example, in case the change at the beginning vertex is increasing the number of upstream paths, the change at the ending vertex will also be increasing the number of upstream paths. A value of -1 implies the changes at the vertices will be opposite to each other. The relationship between the changes and the property value is shown in Equation 5, where the beginning vertex is v1 and the ending vertex is v2. The property value is decided based on the way of the majority of the vehicles that move between the two corresponding street sections. If the path passes through both street sections in the same direction, the property value is 1. Otherwise, the property value is -1. The impact property is key for finding the consequential change at the ending vertex given the change at the beginning vertex. As shown in Equation 6, the considerable change at the ending vertex can be computed based on the property value and the initial change at the beginning vertex.

$$\text{impact}(v1, v2) = \text{change}_{v1} \times \text{change}_{v2} \quad (5)$$
$$\text{change}_{v2} = \text{impact}(v1, v2) \times \text{change}_{v1} \quad (6)$$

When constructing a PDG, it may not be necessary to consider the full path of vehicles due to two reasons. First, the full path of vehicles can consist of a large number of road segments.

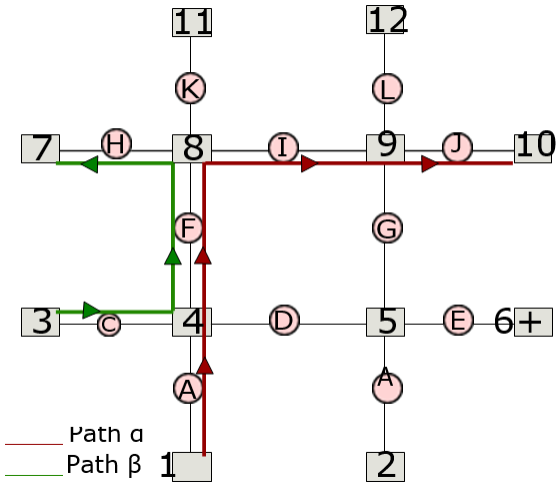
The size of the graph can grow exponentially when the length of path increases. Second, due to the highly dynamic nature of traffic, the coordination of lane-direction changes should only consider the traffic conditions in the future. Therefore, in our implementation, we set an upper limit to the number of street sections in vehicle paths when building a PDG. The limit is called lookup distance in our experiments.

We show an example road network (Figure 6a) and its corresponding PDG (Figure 6b). The road network has 12 roads segments (A to L). There are two paths going through the network, path α and path β. Path α passes through 4 edges (A, B, D, F). Path β passes through 5 edges (A, C, E, D and F). The 7 edges correspond to 6 vertices in the PDG. The PDG contains 2 edges starting from A (A-B, A-C) because path α passes through A, C, E, D and F in the road network. Similarly, the PDG contains 2 edges that starting from A.

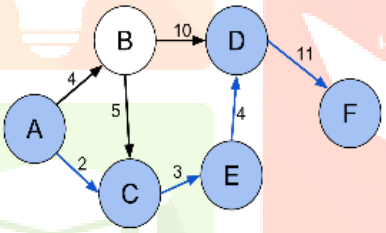
For each edge in the PDG, the value of its impact property is attached to the edge. As path α goes through the edges

(A, B, D and F) in the upstream direction (Figure 6a), the impact value at all the edges between the corresponding vertices is 1 in the PDG (Figure 6b). This is because path β goes through C in the upstream direction but it goes through D in the downstream direction.

The coordinator uses the Direction-Change Assessment algorithm (Algorithm 3) to evaluate the conflicts between path- direction changes. The algorithm traverses through a PDG in a breadth-first manner in iterations. The number of iterations is controlled by a depth parameter (shown as dp in Algorithm 3). In the first round of iteration, the algorithm begins with the lane-direction changes that are proposed by the bottom-layer learning agents. For each vertex with a proposed change, its first-depth



(a) A simple street network with two paths (red and green)



(b) Path dependency graph

neighbours (out-degree nodes) are visited (Step 6).

For each of the neighbours, the considerable change caused by the proposed change is computed. This could be done with the method shown in Equation 6. At that point the algorithm updates the count of conflicts at the neighbour's corresponding street network edge. In another cycle, the algorithm begins with all the neighbour vertices that are visited in the previous round. After each iteration dp is decremented. The algorithm stops when dp reaches zero.

The Direction-Change Evaluation algorithm not only quantifies the conflicts between lane-direction changes but also expands the set of lane-direction changes for the street sections that are visited during the traversal of the PDG. The basis is that the bottom-layer learning agents may not propose path- direction changes for street sections when they don't predict any advantage of the change based on nearby traffic conditions. In any case, the lane-direction changes in other parts of the street network may eventually affect the traffic conditions at these street sections, leading to traffic congestions. The algorithm pre-emptively endeavours lane-direction changes for these street sections when it predicts that there can be considerable changes caused by the changes in other parts of the street network. This will offer assistance in moderating incoming traffic congestions. As shown in Step 6 of Algorithm 3, a Direction-Change Creation algorithm is used for proposing additional lane-direction changes. Details of the Direction-Change Creation algorithm are shown in Algorithm.

Complexity of Coordinating Process. Let us assume there are m number of requests from bottom layer agents. The degree of a node in PDG is deg(v), where $\in^{v} V^{P DG}$. The algorithm traverses m BFTs throughout the PDG for certain depth dp. Then complexity of m BFTs is O(m(deg(v) ^{dp})). However, according to lemma A.1, deg(v) is independent of the road network size or number of paths for a given l. The depth dp is constant irrespective of the road network size. Then the algorithm complexity can be reduced to (m); the algorithm complexity is linear in the number of requests from agents within the buffering period. In the worst case, there can be requests for each road segment of the road network, tt(V, E), leading to the complexity of (OE).

Distributed version. Algorithm 3 can work with a set of distributed agents (coordinating agents) in the upper layer. In Algorithm 3, execution is independent of the order of requests coming from the bottom layer agents. Hence, requests can be processed in a distributed manner. Every coordinating agent traverses first depth and inform changes to other agents. Once every agent finishes their first depth, all coordinating agents begins their second depth, and so on. In such a setting, the complexity of the algorithm is

(1). In this work, we implemented the centralized version, however, when applied to larger street networks, the distributed version can be implemented.

VI. METHODOLOGY

A. We compare the proposed algorithm, CLLA, against DLA and two other baseline algorithms using traffic simulations. We evaluate the performance of the algorithms for street networks based on real traffic information. The impacts of individual parameters of CLLA and DLA are also assessed. The rest of the section points the settings of the experiments. *Experimental setup*

Simulation Model. We simulate a traffic system similar to the ones used for reinforcement learning-based traffic optimization. In our implementation, vehicles on a road link are modelled based on travel time, which is the sum of two values, pure transmit time and waiting time. Pure transmit time is the time taken by a vehicle to travel through the street link at the free-flow speed. Waiting time is the duration that a vehicle waits in a traffic signal queue. When the direction of a path needs to be changed, all existing vehicles in the path need to leave the path and move into the adjacent path in the same direction. Vehicles travelling in the opposite direction can utilize the path only after it is cleared of traffic.

Algorithm 3: Direction-Change Evaluation

Input: $EC_{initial}$, a set of edge-change pairs proposed by the learning agents

Input: tt(V, E), A road network graph. Each edge in the graph has a property, conflict count, which has an integer value that is set to 0 initially.

Input: l, the lookup distance of PDG

Input: dp, the depth of search Output: ECexpanded, a set of edge-change pairs given by the coordinator

(1) Build a PDG based on the next l road segments on the path of vehicles. For each PDG vertex, its properties, **proposed change** and **consequential changes**, are set to empty values initially.
(2) Create an empty set N . For each edge-change pair in $EC_{initial}$, find the corresponding vertex v in PDG and update its proposed change property. Add v to N .
(3) Set the current depth of search to dp.
(4) If the current depth is above 0, do the following steps. Otherwise, jump to Step 8. 5 Create an empty set N J .
(5) For each v in N,
(6) first check whether v has a proposed change. If not, get a proposed change for v using the Direction-Change Creation algorithm. Then for each of v's neighbours at the end of its out-degree arcs, vo, identify the consequential change at the vertex that is caused by the proposed change at v. Add the consequential change to the consequential changes of vo if the change does not exist on the list. If vo already has a proposed change but the proposed change is different to the consequential change at vo, increase the conflict count of the corresponding road network edge by 1. Add vo to N J .
(7) Decrease the current depth of search by *1*. Replace the vertices in N with the vertices in N J . Go back to Step 4.
(8) For Each PDG vertex v with a proposed change, create a corresponding edge-change pair and add the pair to $EA_{expanded}$. Exit the algorithm.

Street Networks. We run experiments based on the real taxi trip information from New York City. The information incorporates the source, the destination and the start time of the taxi trips within the city. We choose a region for simulation since the region contain a larger number of sources and destinations than another region. The street network of the simulation areas is loaded from OpenStreetMap. For a specific taxi trip, the source and the destination are mapped to the nearest OpenStreetMap nodes. The shortest path between the source and destination is calculated. The simulated vehicles follow the shortest paths generated from the taxi trip data.

Comparison baselines. Different to the proposed solution, CLLA, the existing approaches for optimizing lane-directions are based on linear programming, which makes them unsuitable for large-scale dynamic optimization due to the high computation cost. Due to the lack of comparable solutions, we define three baseline solutions, which are used to compare against CLLA. In our experiments, the traffic signals use static timing and phasing, regardless of which solution is used. We conduct comparative tests against the following solutions:

No Lane-direction Allocations (no-LA): This solution does not do any lane-direction change. The traffic is controlled by static traffic signals only.

Demand-based Lane Allocations (DLA): In this solution, the lane-direction changes are computed with Algorithm 1.

Local Lane-direction Allocations (LLA): This solution uses numerous learning agents to decide lane-direction changes. The optimization is performed using the approach depicted in Section V-B. LLA is similar to CLLA but there's no coordination between the agents.

Coordinated Learning-based Lane Allocations (CLLA): This is the two-layer optimization framework depicted in Segment V-A.

B. Evaluation Metrics

We measure the performance of the solutions based on the following metrics.

Average travel time: The travel time of a vehicle is the duration that the vehicle spends on travelling from its source to its destination. We compute the average travel time based on all the vehicles that complete their trips during a simulation. A higher average travel time indicates that the traffic is more congested during the simulation. Our proposed solutions aim to reduce the average travel time. More information about this metric is shown in Section III.

Deviation from free-flow travel time: The free-flow travel time of a vehicle is the shortest possible travel time, achieved when the vehicle travels at the speed limit of the roads without slowing down at traffic lights during its entire trip. Deviation from Free-Flow travel Time (DFFT) is defined as in Equation 7, where ta is the actual time and tf is the free flow travel time. The lowest

value of DFFT is 1, which is also the best value that a vehicle can achieve.

$DFFT = t_a/t_f$ (7)

C. Parameter Sensitivity Testing

We evaluate the impacts of the hyper-parameters of CCLA and DLA, which are directly related to lane-direction changes in the simulation model. To evaluate the impacts of a particular parameter, we run a set of tests varying the value of the parameter (while keeping

Algorithm 4: Direction-Change Creation

Input: v, a PDG vertex that corresponds to an edge in a road network graph. The value of the imbalance property is set to none initially.

Output: change, the proposed lane-direction change For e, which can be 1(upstream), 0(none) and _1(downstream). The default value is 0.

1 $consequential_{up}$: whether the consequential changes at v include one that increases the number of upstrea lanes.

2 $consequential_{down}$: whether the consequential changes at v include one that increases the number of downstream lanes

```
3 if imbalance = upstream then
4     if consequential_up = True and
       consequential_down = False then
5         change ← 1 (change one lane from downstream
           to upstream)
6 if imbalance = downstream then
7     if consequential_up = False and
       consequential_down = True then
8         change ← −1 (change one lane from upstream
           to downstream)
9 if imbalance = none then
10    if (consequential_up = True and downstream has
       more lanes than upstream then
11        change ← 1 (change one lane from downstream
           to upstream)
12    if (consequential_down = True and upstream has
       more lanes than downstream then
13        change ← −1 (change one lane from upstream
           to downstream)
```

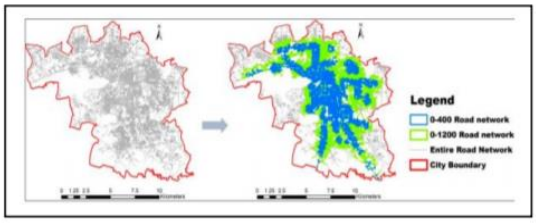


Figure 7: The street network of simulation areas in Bhopal

the value of other parameters at their default detailed in Table I). The average travel time is detailed for each of the tests. The detailed settings of the parameters are shown in Table I. We depict the parameters as follows. **Cost of lane-direction change in CLLA:** The cost of a lane direction change is the time spent on clearing the path that ought to be changed. When the direction of a path changes, all the existing vehicles in the path need to leave the path before the path can be utilized by the vehicles from the opposite direction. The time spent on clearing the path can vary due to different arbitrary components in the real world. For example, the vehicles in the path may not be able to move to an adjacent path instantly in case the adjacent path is highly congested. We vary the value of this parameter in a large range, from 40 seconds to 480 seconds.

Table I: Settings used in the parameter sensitivity experiments

Parameter	Range	Default value
Cost of lane-direction change in CLLA and DLA (seconds)	40-480	120
Aggressiveness of lane-direction change in CLLA (seconds)	100-1000	300
Depth in CLLA	1-5	2
Lookup distance in CLLA	3-5	2
Update period in CLLA (minutes)	0.3-20	0.3
Update period in DLA (minutes)	2.0-20	10

Aggressiveness of lane-direction change in CLLA: This parameter affects the minimum interval between lane-direction changes. A lane-direction change can only happen when there's a traffic imbalance between the two directions at street segment. The imbalance is computed based on the model as shown in Equation 4 (Section V-C). Based on an existing study, we set the threshold percentage P of the model to 65% and require that the traffic imbalance must last for a minimum time period before a lane-direction change can be performed. We define the aggressiveness of lane-direction changes in CLLA as the length of the period. When the period is short, the framework can perform lane-direction changes at smaller intervals, and vice-versa. **Depth in CLLA:** This is the parameter dp used in Algorithm 3. When the depth is larger, CLLA can explore more vertices in the PDG, which permits it to identify the impact of a lane-direction change on the street sections that are further away from the area of the change. **Lookup distance in CLLA:** This is the parameter l used in Algorithm 3. It can affect the number of vertices and the number of edges in a PDG. With a higher lookup distance, the PDG should consider more street sections in the path of vehicles, which can offer assistance in recognizing the impact of lane-direction changes at a longer distance but can increase the size of the graph at the same time.
Update period in CLLA: This parameter controls the frequency at which coordinating agents decide on lane direction changes. CLLA is appropriate for highly dynamic traffic environments. Thus the update period Δt can be set to a low value. We vary the value of this parameter between 0.3 minute to 20 minutes with the default value set to 0.3 minute. **Update period in DLA:** This parameter affects the frequency at which DLA optimizes lane-direction changes. DLA decides on lane-direction changes based on the traffic request that's collected within the update period Δt prior to the optimization process. We vary the value of this parameter between 2.5 min to 20 min with the default value set to 10.

VII. RESULTS

We now display experimental results when comparing CLLA against the baseline algorithms in the first portion, and show the sensitivity analysis to the parameter values of the algorithms in the second part.

Table II: The percentage of vehicles with a DFFT of higher than 10

Solution	Long Island	Midtown Manhattan
DLA	10.03%	49.51%
LLA	7.76%	44.18%
CLLA	7.75%	46.13%

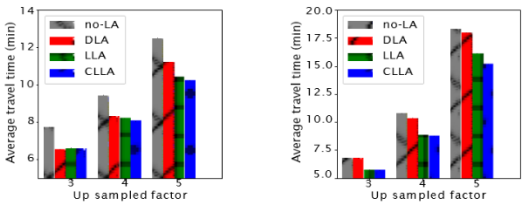
A. Comparison against the baselines

This experiment compares the performance of the four solutions, which are depicted in Section VI-A. We run a number of simulations in this experiment. For each simulation, we extract taxi trip data for one hour using the real taxi trip data from New York. Based on the real data, we create traffic in the simulation. The experiment is done for two regions Figure 8: Performance of four solutions with dynamic traffic as shown in Figure 7a and Figure 7b. To simulate a larger variety of traffic scenarios, we also up-sample the trip data to produce more vehicles. We define an Up Sampled Factor, which is the number of vehicles that are created based on each taxi trip in the taxi data. For LLA and CLLA, the learning rate α is 0.001 and the discount factor used by Q-learning is 0.75. The parameter minLoad of DLA is set to 100. For other parameters of the solutions, we use the default values as shown in Table I.

Average travel time: Figure 8a and Figure 8b show the average travel time accomplished with the four solutions. CLLA beats the other solutions in both simulation regions. We are able to observe that the average travel time of LLA and CLLA is significantly lower compared to the average travel time of no-LA, which shows the advantage of dynamic lane-direction changes. In spite of the fact that DLA achieves lower travel times than no- LA, it does not perform well compared to CLLA for both regions. CLLA performs reliably way better than LLA, because LLA only makes lane-direction changes based on nearby traffic data without coordination. We also test the performance of the solutions for a different situation, where the traffic request is static. Vehicles are produced at a constant rate during a 30-minute period. Under this setting, the traffic is less dynamic than the previous situation, where the traffic request is based on real information. Figure 9a and Figure 9b show the average travel time accomplished with the four solutions. Interestingly, DLA performs as great as CLLA. This is often due to the fact that DLA optimizes traffic based on the evaluated traffic request. As the traffic request is kept constant, the estimated demand can match the actual request, resulting within the great performance of DLA. On the other hand, CLLA is created for highly dynamic traffic situations. When the traffic is static,

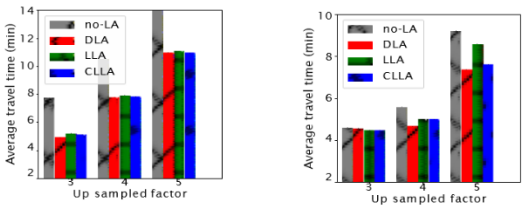
such as in this situation, the advantage of the solution is limited. The results show that DLA can work well with static traffic but does not work well with highly dynamic traffic. CLLA on the other hand works well in both situations: significantly outperforming the baselines in dynamic situations, and matching the performance of DLA in static situations.

Deviation from free-flow travel time (DFFT): Table II shows the percentage of vehicles whose travel time is 10 times or more than their free-flow travel time. The results show that LLA and CLLA are able to achieve a lower deviation from the free-flow travel time compared to DLA.



(a) Long Island (b) Midtown Manhattan

Figure 8: Performance of four solutions with dynamic traffic

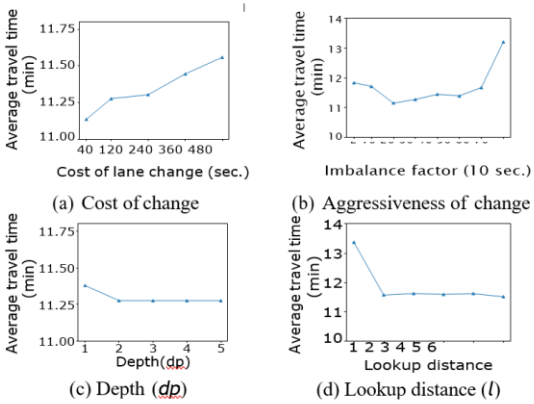


(a) Long Island (b) Midtown Manhattan

Figure 9: Performance of four solutions with static traffic

B. Parameter sensitivity testing

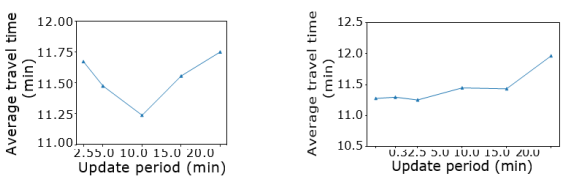
For evaluating the impacts of individual parameters, we run simulations within the region shown in Figure 7a. Each simulation lasts for one hour, during which the traffic is created based on the real taxi trip information from the region. Figure 10 shows the impacts of four parameters of CLLA. Figure 11 compares the impacts of the update period between DLA and CLLA. Our result shows that the travel time increases when the cost of a lane-direction change increases (Figure 10a). The result demonstrates that lane-direction changes may not be beneficial in all circumstances. When the cost of lane-direction changes is high, performing the changes can cause significant interruption to the traffic and invalidate the advantage of the changes. Figure 10b shows how the aggressiveness of lane-direction changes can affect the travel time of vehicles. The result shows that a low level of aggressiveness and a high level of aggressiveness have a negative impact on travel times. When the level of aggressiveness is low, the lane-direction changes can only happen in large intervals. Thus the changes may not adjust to the dynamic change of traffic. When the level of aggressiveness is high, the system changes the direction of paths at a high frequency, which can cause significant interruption to the traffic attributed to taking the time to clear the paths during the changes. Our result shows that the best depth for traversing the PDG is 2 (Figure 10c). When the depth changes from 1 to 2, we observe a decrease in travel time. However, when the depth is higher than 2, we don't observe a decrease of travel time. When the depth is higher, the system can recognize the affect of a lane-direction change that are further away. However, the affect can become negligible in case the lane-direction change is far away. This can be the reason there's no improvement of travel time when the depth is higher than 2. Figure 10d shows that a larger lookup distance can result



(a) Cost of change (b) Aggressiveness of change

(c) Depth (*dp*) (d) Lookup distance (*l*)

Figure 10: Effects of four parameters of CLLA



(a) DLA update period (b) CLLA update period

Figure 11: Effects of the update period of CLLA and DLA

in a lower average travel time. When the lookup distance increases, CLLA considers more street sections in a vehicle path when building the PDG. This helps recognize the significant lane-direction changes on the same path. Decrease in the average travel time becomes less significant when the lookup distance is higher than 2. This is because the impact of a path- direction change diminishes when the change is further away. When the update period Δt of DLA is below 5 minutes or past 15 minutes, it is less likely to get a great estimation of traffic request, which can lead to a relatively high travel time (Figure 11a). The average travel time is at its minimum when Δt is set to 10 minutes. Different to DLA, the travel time accomplished with CLLA develops gradually with the increase of Δt until Δt reaches past 15 minutes. The moderately steady performance of CLLA shows that the coordination between lane-direction changes can offer assistance in relieving traffic congestion for a certain period of time within the future. In case minimizing the average travel time is of priority, one can set Δt to a very low value, e.g., 5 minutes. If one needs to decrease the computation cost of the optimization while accomplishing a reasonably good travel time, the Δt can be set to a bigger value, e.g., 15 minutes.

CONCLUSION

The study has shown that efficient traffic optimization can be accomplished with dynamic lane-direction configurations. The proposed hierarchical multi-agent solution, CLLA, can offer assistance in decreasing travel time by combining machine learning and the global coordination of lane-direction changes. The proposed solution adjusts to significant changes of traffic requests in a timely manner, making it a practical choice for realizing the potential of connected AV in traffic.

REFERENCES

[1] L. R. Ford and D. R. Fulkerson, “Maximal flow through a network,” in Classic papers in combinatorics. Springer, 2009, pp. 243–248.
[2] L. Fleischer and M. Skutella, “Quickest flows over time,” SIAM J.
[3] B. Wolshon and L. Lambert, “Planning and operational practices for reversible roadways,” Institute of Transportation Engineers. ITE Journal, vol. 76, no. 8, pp. 38–43, August 2006.
[4] L. Lambert and B. Wolshon, “Characterization and comparison of traffic flow on reversible roadways,” Journal of Advanced Transportation, vol. 44, no. 2, pp. 113–122, 2010.
[5] J. J. Wu, H. J. Sun, Z. Y. Gao, and H. Z. Zhang, “Reversible lane-based traffic network optimization with an advanced traveller information system,” Engineering Optimization, vol. 41, no. 1, pp. 87–97, 2009.
[6] K. F. Chu, A. Y. S. Lam, and V. O. K. Li, “Dynamic lane reversal routing and scheduling for connected autonomous vehicles,” in 2017 International Smart Cities Conference (ISC2), Sep. 2017, pp. 1–6.
[7] M. Hausknecht, T. Au, P. Stone, D. Fajardo, and T. Waller, “Dynamic lane reversal in traffic management,” in 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC), Oct 2011, pp. 1929–1934.
[8] Georgia Department of Transportation, “Advantages and disadvantages of reversible managed lanes,” http://www.dot.ga.gov/BuildSmart/ Studies/ManagedLanesDocuments/Emerging%20Issues-Reversible% 20Managed%20Lanes.pdf, 2010.
[9] R. K. Narla, T. Technology, and U. States, “The Evolution of Connected Vehicle Technology : From Smart Drivers to Smart Cars to . . . Self- Driving Cars,” no. July, pp. 22–26, 2013.
[10] K. Ramamohanarao, J. Qi, E. Tanin, and S. Motallebi, “From how to where: Traffic optimization in the era of automated vehicles,” in Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ser. SIGSPATIAL ’17. New York, NY, USA: ACM, 2017, pp. 10:1–10:4.
[11] R. S. Sutton and A. G. Barto, Introduction to Reinforcement Learning, 1st ed. Cambridge, MA, USA: MIT Press, 1998.
[12] L. R. Ford and D. R. Fulkerson, “Constructing maximal dynamic flows from static flows,” Oper. Res., vol. 6, no. 3, pp. 419–433, jun 1958.
[13] E. Ko¨hler, R. H. Mo¨hring, and M. Skutella, Traffic Networks and Flows over Time. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 166–196.
[14] N. R. Ravishankar and M. V. Vijayakumar, “Reinforcement learning algorithms: Survey and classification,” Indian Journal of Science and Technology, vol. 10, no. 1, 2017.
[15] E. Walraven, M. T. Spaan, and B. Bakker, “Traffic flow optimization: A reinforcement learning approach,” Engineering Applications of Artificial Intelligence, vol. 52, pp. 203–212, 2016.
[16] K.-L. A. Yau, J. Qadir, H. L. Khoo, M. H. Ling, and P. Komisarczuk, “A Survey on Reinforcement Learning Models and Algorithms for Traffic Signal Control,” ACM Computing Surveys, vol. 50, no. 3, pp. 1–38, 2017.
[17] P. Mannion, J. Duggan, and E. Howley, “An Experimental Review of Reinforcement Learning Algorithms for Adaptive Traffic Signal Control,” pp. 47–66.
[18] M. Aslani, S. Seipel, M. S. Mesgari, and M. Wiering, “Traffic signal optimization through discrete and continuous reinforcement learning with robustness analysis in downtown Tehran,” Advanced Engineering Informatics, vol. 38, no. June 2017, pp. 639–655, 2018.