2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

Route Coverage Testing for Autonomous Vehicles via Map Modeling

Yun Tang, Yuan Zhou, Fenghua Wu, Yang Liu, Jun Sun, Wuling Huang, and Gang Wang

Abstract— **Autonomous vehicles (AVs) play an important role in transforming our transportation systems and relieving traffic congestion. To guarantee their safety, AVs must be sufficiently tested before they are deployed to public roads. Existing testing often focuses on AVs' collision avoidance on a given route. There is little work on the systematic testing for AVs' route planning and tracking on a map. In this paper, we propose CROUTE, a novel testing method based on a new AV testing criterion called route coverage. First, the map is modeled as a labeled Petri net, where roads, junctions, and traffic signs are modeled as places, transitions, and labels, respectively. Second, based on the Petri net, we define junctions' topology features and route features for junction classification. The topology feature describes the topology of roads forming the junction, and the route feature identifies the actions that a vehicle can take to follow a route. They can characterize route types on a map. Hence, route coverage measures how many route types are covered. We then propose a systematic method that aims to cover all route types for a well-designed AV system with a small number of test cases. We implement and evaluate CROUTE on Baidu Apollo running with the LGSVL simulator. We carry out testing on the map from a section of *San Francisco* and find six different types of issues in Apollo. The experiment results show the validity of route coverage and the efficiency of CROUTE.**

I. INTRODUCTION

Autonomous vehicles (AVs) play an essential role in relieving traffic congestion and eliminating accidents in intelligent transportation systems. Many companies have been devoting themselves to this domain, such as Google Waymo [1], Baidu Apollo [2], and Autoware [3]. Before an AV is deployed to the real world, its autonomous driving system (ADS) must be tested sufficiently. However, ADS testing is challenging due to the open and unpredictable environments.

Researchers have proposed different techniques for critical scenario generation [4]–[17]. The criticality of scenarios can be evaluated in terms of safety, functionality, mobility, and rider's comfort [4]. Most of the existing work focuses on ADSs' collision avoidance and introduces different safety-related metrics, such as accident rate [10], drivable area [11], [12], time-to-collision [13]–[15], and performance boundary [16], [17]. There is little work on systematic testing of AVs' route planning and tracking function on a map, which is important and necessary. First, heuristic search methods, such as Dijkstra's algorithm and A* algorithm [18], are widely applied for route planning in the current ADSs. They require some predefined parameters to compute costs of paths in the directed graphs derived from uploaded maps. We thus must test whether the parameters are suitable for different maps and situations. Second, during real-time route tracking, AVs encounter different road conditions that may block the previously computed route. In this case, we need to test whether the system can re-route successfully.

Given the above testing requirements, in this paper, we propose a metric to identify route diversity and a new method, CROUTE, to test the performance of ADSs' route planning and tracking systematically on a map. CROUTE implements a new AV testing criterion called route coverage. We first model the map as a labeled Petri net. Each place represents a road; each transition is a set of junction lanes connecting the same roads; the traffic signs at junctions are described as labels assigned to transitions. Second, based on the Petri net, we propose *route type* to measure route diversity on the map. It combines junction topology features, which describe the topology and connection relations of the roads at a junction, and route features, which describe the sequence of actions an AV needs to take to track a route. Hence, CROUTE aims to test as many route types as possible. Third, a systematic test case generation method is proposed for iteratively achieving a higher route coverage. Our method can generate a small number of test cases to cover all route types on a map for a well-designed system. We implement and evaluate CROUTE on *Apollo*, running with the LGSVL simulator on the map of a section of *San Francisco*. To avoid uncertainties in the perception module, we use ground truth perception data from LGSVL. CROUTE discovers six different types of issues, all of which are confirmed by the development team. We also compare CROUTE with complete route testing and random testing. The results show the effectiveness and efficiency of CROUTE.

The contributions of this paper are as follows.

- We propose a measure to characterize route diversity in a map and a testing criterion for route coverage testing on AV's route planning and tracking functionality.
- We propose a systematic test case generation method on a map, which can generate a small set of test cases with 100% route coverage in a well-designed ADS.

This work was supported in part by Alibaba Group through Alibaba Innovative Research Program and Alibaba-NTU Joint Research Institute, the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant (19-C220-SMU-002) and Tier 2 grant (MOE-T2EP20120-0004), the Singapore National Research Foundation (NRF) under its National Cybersecurity R&D Program (Award Nos. NRF2018NCR-NCR005-0001 and NRF2018NCR-NSOE003-0001), and the NRF Investigatorship NRFI06-2020-0022. *(Corresponding author: Yuan Zhou)*

Yun Tang is with Alibaba-NTU Joint Research Institute, Nanyang Technological University, Singapore. (E-mail: yun005@e.ntu.edu.sg)

Yuan Zhou, Fengua Wu and Yang Liu are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. (E-mail: {y.zhou, fenghua.wu, yangliu}@ntu.edu.sg)

Jun Sun is with the School of Computing and Information Systems, Singapore Management University, Singapore. (E-mail: junsun@smu.edu.sg)

Wuling Huang and Gang Wang are with Alibaba DAMO Academy, Alibaba Group, China. (E-mail: wuling.hwl@cainiao.com, wg134231@alibaba-inc.com)

- We implement the route coverage testing method and perform global testing on *Apollo*, one of the state-of-the-art ADSs. Real issues are discovered, which have been confirmed by the development team.

II. Related Work

Existing AV testing can be divided into two types: on-road testing and simulator-based testing. An extensive and rigorous road test for AVs is necessary but often risky, costly, and insufficient. We cannot test on every road and every variable that could affect AV safety. Hence, before on-road testing, developers would like to test AVs via simulators.

Even though we can generate various scenarios in a simulator, the number of scenarios increases exponentially with the dimension of parameters forming a scenario. Hence, researchers propose different methods to generate critical scenarios [4]–[17]. For example, some methods have been proposed to extract collision scenarios from off-the-shelf datasets such as crash reports and human driving data [5]–[7]. Deep learning technologies can also be used to increase data diversity in these datasets [8]; in [9], collision scenarios are replicated from dashcam crash videos. Based on some predefined metrics, such as accident rate [10], drivable area [11], [12], time-to-collision [13]–[15], and performance boundary [16], [17]. However, most of them focus on the generation of critical scenarios for collision testing. There is little work on route planning and tracking testing of AVs.

Route planning and tracking is one of the basic functionalities for AVs. Route planning aims to generate a route/path from the initial position to the target, while route tracking determines how to move along the planned route. González *et al* summarize motion planning algorithms for AVs [18]. Among these algorithms, heuristic search methods, such as Dijkstra's algorithm [19], A* based algorithms [20], and RRT-based algorithms [21], are widely used to search for a feasible route. For example, in Apollo and Autoware, A* algorithm is implemented for route planning. It needs to abstract a direct graph from a map and assign some predefined parameters to the graph's nodes and edges for cost computation. These parameters may show some bias and forbid some feasible routes. For route tracking, different methods, such as Frenét Frame method [22], optimization [23]–[25], and behavior planning [26], [27], have been proposed to produce actions to follow the generated route. However, due to real-time changes in road conditions, the generated route may be blocked and become unfeasible. A new route should be re-planned. Hence, it is important to test whether an ADS can cover all possible routes under different conditions on a map. As far as we know, there is no work on route coverage testing for ADSs on a complete map.

III. Problem Statement

Modern AVs operate under the guidance of high-definition (HD) maps, which usually contain map geometry and semantic objects such as lane boundaries, intersections, crosswalks, parking spots, stop signs, and traffic lights [28]. Lanes are the atomic geometry in a map and describe the paths that a vehicle can move along. A *lane* is bounded by two lane boundaries and contains a reference path set as its centerline. A vehicle should move along the reference path in a lane. In the following description, we use its reference path to represent a lane. A *road* is a set of lanes with the same direction, among which a vehicle can perform lane changing at any position. A *junction* contains a set of junction lanes that connect different roads. Vehicles cannot change lanes at a junction. For example, Fig. 1 shows a junction connecting 8 roads. In this example, the bold and the dashed lines are lane boundaries, while the bold lines are also road boundaries; the filled regions are crosswalks; the arrow lines are the reference paths of the corresponding lanes, e.g., l_2 is a junction lane starting from a and ending at b, and l_{17} and l_{18} are two road lanes forming road R_1; R_1 - R_8 are eight roads connected by the junction, and each one contains two lanes; the junction is controlled by traffic lights.

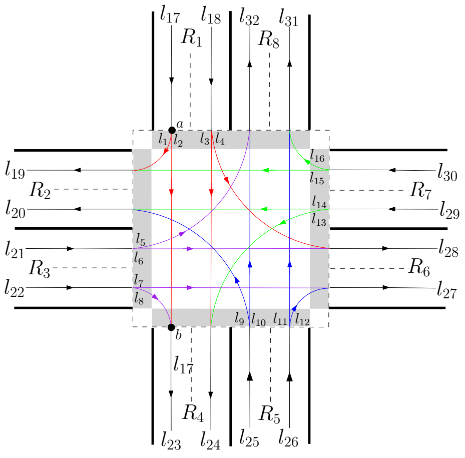


Fig. 1: A junction J_0 in *San Francisco*.

Given a map, the ADS plans a route to a destination on the map. There are various routes to the destination since the AV can select different junction lanes. Whenever obstacles block a route, the AV should select another one unless all routes are blocked. It is important to test the AV's capacity to plan and track different routes on the map under different situations. Hence, our problem can be stated as follows:

Problem: *Given an ADS and a map, design a method to evaluate the system's route planning and tracking functionality on the map by testing on a small number of routes.*

To plan a global route on a map in any situation, the ADS should be able to cover every route at a junction. Hence, we focus on route planning and tracking testing at each junction of the map. Before describing the solution, we state some assumptions and scopes considered in this paper.

- An ADS can be roughly divided into two parts: perception and motion. In this paper, we bypass the perception part and focus on the motion part.
- Since we focus on the AV's ability to cover all possible routes under different situations, we simplify collision avoidance by only adding static obstacles.
- The length of a road supports continual lane changing from the leftmost to the rightmost lanes, and vice versa. It should be satisfied during the design of the road.

IV. Methodology

A. Map Modeling

Let l_i be a lane on a map. A road R_r containing n_r lanes, say $l_1^r, \ldots, l_{n_r}^r$, is denoted as $R_r = l_1^r \| \ldots \| l_{n_r}^r$, and the set of all roads is denoted as $R = \{R_1, R_2, \ldots, R_n\}$. A junction J_j containing m_j lanes is denoted as $J_j = \{l_1^j, \ldots, l_{m_j}^j\}$, and the sets of junctions and junction lanes are denoted as $J = \{J_1, \ldots, J_m\}$ and $L^J = \bigcup_{j \in \mathbb{N}_m} J_j$, respectively, where $\mathbb{N}_m = \{1, 2, \ldots, m\}$. Given a junction lane l_i^j, $l_i^j = (l^{r_1}, l^{r_2})$ if l^{r_1} and l^{r_2} are connected directly by l_i^j, where $l^{r_1} \in R_{r_1}$, and $l^{r_2} \in R_{r_2}$; R_{r_1} and R_{r_2} are called the incoming and the outgoing roads of l_i^j, denoted as $I(l_i^j)$ and $O(l_i^j)$, respectively. The traffic signs related to a junction are stop sign (sp), traffic light (lt), entering crosswalk (cw_I), and exiting crosswalk (cw_O).

We apply a labeled Petri net to describe a map. A labeled Petri net is a tuple $\langle S, T, A, \Sigma, \lambda \rangle$, where S and T are finite sets of places and transitions, respectively, $A \subseteq (S \times T) \cup (T \times S)$ is a set of arcs connecting places and transitions, Σ is a set of labels, and $\lambda : T \to \Sigma$ is a labeling function assigning a label to each transition. $\forall e \in S \cup T$, $Pre(e)$ and $Post(e)$ denote the preceding and the succeeding sets of e, respectively. Map modeling can be described as follows. (1) Each road is modeled as a place, i.e., $S = \{R_1, R_2, \ldots, R_n\}$. (2) For any two roads R_{r_1} and R_{r_2}, the set $\{(l_{i1}^{r_1}, l_{i2}^{r_2}) : l_{i1}^{r_1} \in R_{r_1}, l_{i2}^{r_2} \in R_{r_2}\}$ is modeled as a transition T_i^j if it is nonempty; R_{r_1} and R_{r_2} are called incoming and outgoing roads of T_i^j, respectively. Hence, each junction J_j is modeled as a set of transitions, denoted as $T^j = \{T_1^j, T_2^j, \ldots, T_{j_k}^j\}$, and we have $T = \bigcup_{j \in \mathbb{N}_m} T^j$. (3) Given a place R_r and a transition T_i^j, $(R_r, T_i^j) \in A$ if R_r is the incoming road of T_i^j, and $(T_i^j, R_r) \in A$ if R_r is the outgoing road of T_i^j. The set of roads connected by J_j is $R^j = R_{\mathrm{I}}^j \cup R_{\mathrm{O}}^j$, where $R_{\mathrm{I}}^j = \bigcup_{T_i^j \in T^j} Pre(T_i^j)$ and $R_{\mathrm{O}}^j = \bigcup_{T_i^j \in T^j} Post(T_i^j)$ are the incoming and outgoing roads of J_j, respectively. (4) The set of labels is $\Sigma = \{a = (sp, lt, cw_I, cw_O) : a \in \{0, 1\}^4\}$, where $sp = 1$ means T_i^j is controlled by stop signs, $lt = 1$ means T_i^j is controlled by traffic lights, and $cw_I = 1$ (resp., $cw_O = 1$) denotes the crosswalk between T_i^j and its incoming (resp., outgoing) road. Fig. 2 is the Petri net for J_0 shown in Fig. 1, where $\forall i \in \mathbb{N}_{12}$, $\lambda(T_i^0) = (0, 1, 1, 1)$.

Remark 1: Since each junction lane can connect one and only one incoming/outgoing road, the generated Petri net is a state machine, i.e., $\forall T_i^j \in T$, $|Pre(T_i^j)| = |Post(T_i^j)| = 1$.

B. Route Diversity and Route Coverage

Based on the labeled Petri net, we first quantify the diversity of routes. It is measured by the junction topology and route features at a junction. Junction topology characterizes the topology of roads forming the junction, while route features reflect different motion modes to cross a junction.

1) Junction Topology Feature: Given a junction J_j, let CR^j be a function mapping each road to its connecting roads at J_j: $\forall R_{r_i} \in R_j^{\mathrm{I}}$, $CR^j(R_{r_i}) = Post(Post(R_{r_i}))$; $\forall R_{r_i} \in R_j^{\mathrm{O}}$, $CR^j(R_{r_i}) = Pre(Pre(R_{r_i}))$. Starting from R_{r_1}, suppose the counter-clockwise road sequence in R^j is

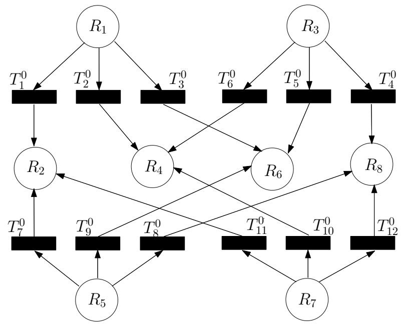


Fig. 2: Petri net for the junction shown in Fig. 1

$C_{r_1}^j = R_{r_1} R_{r_2} \ldots R_{r_k}$, where $k = |R^j|$. The road indexes in $C_{r_1}^j$, denoted as $ID_{r_1}^j$, is determined as follows: If $R_{r_i} \in R_j^{\mathrm{I}}$, $ID_{r_1}^j(R_{r_i}) = i$; otherwise, $ID_{r_1}^j(R_{r_i}) = -i$.

Definition 1: Given a junction J_j and $R_{r_1} \in R^j$, the *topology feature* of J_j with respect to R_{r_1} is defined as a vector $TV_{r_1}^j$ such that $TV_{r_1}^j(R_{r_i}) = ID_{r_1}^j(CR^j(R_{r_i}))$.

Definition 2: Given a junction J_j and $R_{r_1} \in R^j$, the *weak topology feature* of J_j with respect to R_{r_1} is defined as a vector $wTV_{r_1}^j$ such that $\forall R_r \in R_j^{\mathrm{I}}$, $wTV_{r_1}^j(R_r) = |Post(R_r)|$, and $\forall R_r \in R_j^{\mathrm{O}}$, $wTV_{r_1}^j(R_r) = -|Pre(R_r)|$.

For example, in Fig. 1, we have $CR^o(R_1) = \{R_2, R_4, R_6\}$ and $CR^o(R_2) = \{R_1, R_5, R_7\}$. For the road sequence $C_1^0 = R_1 R_2 \ldots R_8$, we have $TV_1^0(R_1) = \{-2, -4, -6\}$ and $wTV_1^0(R_1) = 3$, and $TV_1^0(R_2) = \{1, 5, 7\}$ and $wTV_1^0(R_2) = -3$.

The topology feature describes the relative locations and connection relations of the roads forming a junction. In contrast, the weak topology feature only describes the number of roads that a road can connect at the junction. Since the topology feature is related to a reference road, a junction contains multiple topology features.

Definition 3: Two junctions J_{j_1} and J_{j_2} are topology equivalent if $\forall R_{r_1} \in R^{j_1}$, there exists a road $R_{r_2} \in R^{j_2}$ such that $TV_{j_1}(R_{r_1}) = TV_{j_2}(R_{r_2})$.

To determine all topology equivalence classes of a map, we need to compute and compare all junctions' possible topology features. It may be time-consuming. So, we first compute the set of weak topology features of a junction since we only need to compute the weak topology feature related to one road. All of its rotations form the set of weak topology features. Then, we only need to compare the topology features of the junctions whose weak topology features are the same.

2) Route Feature: The routes may vary at the junctions with the same topology feature due to different traffic signs and road lanes. They can generate different motion actions, e.g., lane following (including over-taking), lane changing, crosswalk passing, stop sign waiting, traffic light obeying, etc. In the sequel, we define route features to describe the ego vehicle's actions to follow the selected road.

Given a road pair $R_{r_1} T_i^j R_{r_2}$ in J_j, where $R_{r_1} = Pre(T_i^j)$ and $R_{r_2} = Post(T_i^j)$, a *route* of J_j in the road pair, denoted as p, consists of a sequence of lanes in R_{r_1}, a junction lane in T_i^j, and a sequence of lanes in R_{r_2}. Let $|\alpha(p)|$ be the number of covered lanes in R_{r_1} to enter J_j, and $|\beta(p)|$ be the number of covered lanes in R_{r_2} to reach the destination, where $\alpha(p)$ or $\beta(p)$ is positive for right lane changing and negative for left lane changing along the lane direction.

Definition 4: Given a junction J_j, suppose p is a route in $R_{r_1}T_i^jR_{r_2}$, the *route feature* of p, denoted as $F(p)$, is a vector $(\alpha(p), \beta(p), \gamma(p))$, where $\gamma(p) = \lambda(T_i^j)$. The route features of all possible routes in the junction is denoted as $F(J_j) = \bigcup_p \{F(p)\}$.

For example, as shown in Fig. 1, for route $p = l_{17}l_{18}l_3l_{24}$, we have $F(p) = (-1, 0, (0, 1, 1, 1))$.

The topology feature affects the high-level road selection at a junction, while the route feature determines the vehicle's low-level actions to move along the selected road. Hence, they can measure route diversity.

Definition 5: The type of a route at a junction is the combination of the junction's topology feature and the route's route feature.

Definition 6: Two junctions are route equivalent if they contain the same route types.

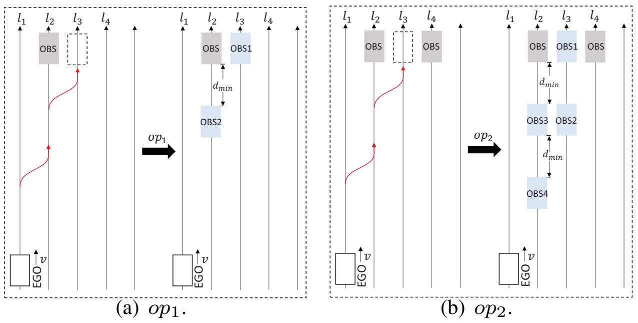
Based on Definition 6, the junctions on a map can be divided into different junction classes, denoted as $\{\mathcal{J}_k\}$, such that the junctions in the same class are route equivalent. $\forall J_j \in \mathcal{J}_k$, $F(\mathcal{J}_k) = F(J_j)$. $\mathcal{J}_1$ is a superset of $\mathcal{J}_2$ if they are topology equivalent and $F(\mathcal{J}_1) \supsetneqq F(\mathcal{J}_2)$. The route types of a junction are subsumed by its superset. We can improve the diversity of routes by focusing on only the supersets.

The detailed process of junction classification on a map can be summarized as follows. (1) Junction topology classification: For any two junctions J_{j_1} and J_{j_2}, $\forall r_1 \in R^{j_1}$, if $\exists r_2 \in R^{j_2}$ such that $wTV_{r_1}^{j_1} = wTV_{r_2}^{j_2}$, then determine whether there exist a road $r_3 \in R^{j_2}$ such that $TV_{r_1}^{j_1} = TV_{r_3}^{j_2}$; if "yes", J_{j_1} and J_{j_2} are grouped into one class. (2) Junction classification: For the junctions in each topology class, compute its route types and classify the junctions with the same route types in the same class, say $\mathcal{J}_k$. Hence, we can finally obtain the junction classes $\{\mathcal{J}_k\}$.

C. Test Case Generation for Route Coverage Testing

The parameters affecting route planning are the initial and target positions of the ego vehicle and the positions of obstacles. Without loss of generality, we set the ego vehicle's initial and target positions as the start of the incoming road lane and the end of the outgoing road lane. Hence, the format of test cases in our testing is a vector containing the ego vehicle's initial and target lanes and the obstacles' positions.

Since routes in different classes have different types, we first select a junction in each class. The selection process is as follows. (1) Since all vehicles require a certain distance to perform lane changing, we first select the junctions with longer incoming and outgoing roads. (2) Among the selected junctions, we select junctions with wider lane widths since lane width affects lane changing. (3) If there are multiple selected junctions, we further select the junctions that have the maximal number of junction lanes since such junctions contain more routes and may be able to discover more individual-difference issues. (4) Finally, a random selection is performed if there are still multiple selected junctions.



(a) op_1. (b) op_2.

Fig. 3: Two mutation operators with $k = 3$ and $k_0 = 1$.

For each selected junction J_j, we first generate all base test cases, i.e., test cases without any obstacle. $\forall T_i^j \in T^j$, let $R_{r_1} = Pre(T_i^j)$ and $R_{r_2} = Post(T_i^j)$, then $\forall l_{i1}^{r_1} \in R_{r_1}$ and $\forall l_{i2}^{r_2} \in R_{r_2}$, $(l_{i1}^{r_1}, l_{i2}^{r_2}, \emptyset)$ is a base test case, where $l_{i1}^{r_1}$ means that the initial position of the ego vehicle is the start of $l_{i1}^{r_1}$, $l_{i2}^{r_2}$ means that the target position of the ego vehicle is the end of $l_{i2}^{r_2}$, and $\emptyset$ means that there are no obstacles around. All such combinations of road lanes at junction J_j form the set of base test cases.

Second, we search for new test cases by adding static obstacles. Given a base test case $(l_{i1}^{r_1}, l_{i2}^{r_2}, \emptyset)$, the maximal number of different routes connecting $l_{i1}^{r_1}$ and $l_{i2}^{r_2}$ is $|T_i^j|$. If $|T_i^j| > 1$, we generate new test cases by adding obstacles to block the current route. For any route p having the same initial and target lanes with the base test case, $\alpha(p)+\beta(p)$ is a constant. Hence, our mutation can focus on the motion in the incoming road. Suppose during the execution of the current test case, the ego vehicle goes through $l_1, l_2, \ldots, l_k$ sequentially and enters into the junction, where $l_{k-1}, \ldots, l_{k-k_0}$ contain obstacles, and l_{k-k_0-1} does not contain any obstacle. l_{k+1} is the next lane of l_k along the current lane changing direction. To block the current route, we introduce two operators: (1) op_1: if l_{k+1} does not contain any obstacles, add an obstacle on each of $l_{k-k_0}, \ldots, l_k$, as shown in Fig. 3(a), and (2) op_2: if l_{k+1} contains o obstacles, add $o+1$ obstacles to each of $l_{k-k_0}, \ldots, l_k$, as shown in Fig. 3(b). For each lane, the distance between two adjacent obstacles is equal to the minimal lane changing distance. Hence, given a base test case tc_0, we can generate test cases: $tc_0 \xrightarrow{\delta_1} tc_1 \xrightarrow{\delta_2} \ldots \xrightarrow{\delta_k} tc_k$, where $\delta_k \in \{op_1, op_2\}$ and $k \in \{1, \ldots, |T_i^j| - 1\}$.

Algorithm 1 shows the test case generation for a selected junction. First, we run each base test case tc_0 (Lines 2 - 5) and compute the route feature it may cover (Line 6). Then, we compute the route features of all possible routes connecting the initial and target positions in the base test case and check whether they have been covered. If there are no new route features, we do not mutate this test case (Lines 8 and 9). Otherwise, we perform a sequence of mutation operators (Lines 11 - 22). The mutation is terminated when no new route features will be covered in the rest routes (Lines 12 and 13), or the execution of a test case fails, e.g., performing lane changing among junction lanes or failures in reaching the target (Lines 17 and 18).

Theorem 1: For a well-designed self-driving system, the test cases generated by Algorithm 1 can cover all route

Algorithm 1: Route coverage guided testing.

```
Input: Petri net of a junction $J_j$; $R^j = R^j_{\mathrm{I}} \cup R^j_{\mathrm{O}}$ and
       $\mathcal{T}^j = \{T^j_i\}$; the set of route features $F(J_j)$.
1  Initialization: the set of test cases $TC = \emptyset$, the set of
   covered route features $Cov = \emptyset$;
2  for $T^j_i \in \mathcal{T}_j$ do
3      for $\forall(l^{r_1}_{i1}, l^{r_2}_{i2}) \in Pre(T^j_i) \times Post(T^j_i)$ do
4          Generate a base test case $tc_0 = (l^{r_1}_{i1}, l^{r_2}_{i2}, \emptyset)$;
5          Run $tc_0$ and compute the route feature $F(tc_0)$;
6          $TC = TC \bigcup \{tc_0\}$, $Cov = Cov \cup \{F(tc_0)\}$;
7          Compute route features for routes having the same
           initial and target positions with $tc_0$: $PF(tc_0)$ ;
8          if $PF(tc_0) \subseteq Cov$ then
9              continue
10         else
11             for $k = 1 : |T^j_i| - 1$ do
12                 if $PF(tc_0) \subseteq Cov$ then
13                     break;
14                 else
15                     Generate a new test case $tc_k$ from
                       $tc_{k-1}$ with $op_1$ or $op_2$;
16                     Run $tc_k$; $TC = TC \bigcup \{tc_k\}$;
17                     if The execution of $tc_k$ fails then
18                         break
19                     else if $F(tc_k) \in Cov$ then
20                         continue
21                     else
22                         $Cov = Cov \cup \{F(tc_k)\}$ and
                           $PF(tc_0) = PF(tc_0) \setminus \{F(tc_k)\}$
```

features in a junction. Thus, the test cases generated from all selected junctions can cover all route features on a map.

Proof Sketch: On the one hand, by adding an obstacle to the end of the lane where the ego vehicle enters the junction, we can prevent the ego vehicle from passing through the junction with the same junction lane. On the other hand, by adding obstacles to lanes containing obstacles, the ego vehicle can start lane changing early. So it can pass through a sequence of continuously blocked lanes via continual lane changing. Hence, one mutation blocks only one feasible route. By blocking routes one by one, Algorithm 1 can generate test cases to cover all routes at a junction if the system is well-designed. Moreover, Algorithm 1 will generate a small number of test cases since (1) with junction classification, we only need to test one junction in each class, (2) based on Lines 7 - 9, only the base test cases that contain new route features are mutated, and (3) each test case will cover a new route at the junction. ■

V. Experiments

Our experiments are to answer the following questions.

RQ1: Effectiveness. Can CROUTE discover issues in an ADS?

RQ2: Reasonability. Is it reasonable to classify junctions based on route types?

RQ3: Efficiency. Can CROUTE facilitate the testing process?

A. Experiment Setup

We implement and evaluate CROUTE on one of the state-of-the-art ADSs, Apollo 5.0 (LGSVL forked version) [29], which is running on the LGSVL simulator release 2020.01 [30]. The map is a section of *San Francisco* [31]. The software runs on a Ubuntu 18.04 desktop equipped with 2 GTX 1080 Ti GPUs in SLI-mode, Intel Core i7-6850K (6 cores, 12 threads), and 32 GB RAM. To guarantee the correctness of perception data, we bypass the perception module and directly use the ground truth perception data. To save time, each test case is terminated once the ego vehicle reaches the destination lane.

B. RQ1: Effectiveness of CROUTE

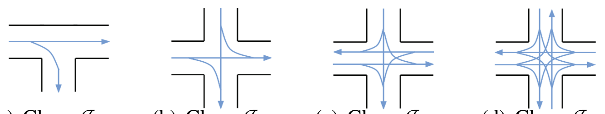
With CROUTE, we discovered the following issues, whose videos and detailed analysis can be found at `https://av-testing.github.io/croute/`.

1) Insufficient distance for continual lane changing: Apollo requires a long distance for lane changing (at least about 50 meters from our experimental discovery), which sometimes is too conservative, resulting in a big detour or even incompleteness of a motion task.
2) Inefficient routing: The default configurations for Apollo's routing algorithm (i.e., A*) prefer to avoid changing lanes before entering a junction even if there are sufficient distances for continual lane changing, resulting in a big detour without any lane changing.
3) Stuck motion at stop signs: When Apollo passes a junction controlled by stop signs, it stops before the white stopping line and never moves again.
4) Insufficient re-routing: When there are obstacles ahead, Apollo always performs overtaking rather than executing lane changing and re-routing a new route. It will result in long-distance motion along the boundary of two adjacent lanes or lane changing in a junction.
5) Lane gap: Apollo has a default configuration to define the maximal width of a lane. When a lane is wider than its configured width, Apollo cannot recognize the actual width but increase the gap with the adjacent lane. Hence, Apollo can plan a feasible route but cannot perform a successful lane changing (e.g., terminating lane changing halfway and turning back to the original one).
6) Map error: Since the HD maps are constructed manually, different errors, such as location mismatch of curbs and disordered points forming a lane, are present.

Among the above issues, 1), 2), 4), and 5) frequently occur during our experiments. Moreover, the last issue is specific to the tested map, and we can do nothing but search for each position of the map. In conclusion, with CROUTE, we can find different issues.

C. RQ2: Reasonability of Junction Classification

We first investigate the generated junction classes for the map. There are 84 junctions in the map. Based on Definition 6, CROUTE divides them into 29 junction classes with 19 unique weak topology vectors. The junctions in the same class are with high similarity. Fig. 4 shows the topology



(a) Class $\mathcal{J}_{46}$. (b) Class $\mathcal{J}_{47}$. (c) Class $\mathcal{J}_{22}$. (d) Class $\mathcal{J}_{3}$.

Fig. 4: The topology diagrams of some junction groups.

TABLE I: Comparison of CROUTE and CRT

	CRT	CROUTE	Random
# Junctions	84	22	22
Test Case Failure Rate	$\frac{848+5330}{9026}$	$\frac{178+249}{1117}$	$\frac{218.49+236.51}{1202.03}$
Route Coverage Rate	293/634	236/634	141.84/323.48
# Issues	7	6	4.65

diagrams of four typical junction classes. The detailed classes and their contained junctions are given at `https://av-testing.github.io/croute/`.

Second, we compare CROUTE with Complete Route Testing (CRT) in terms of the discovered issues. We want the ego vehicle to go through every route in the map for CRT. In CROUTE, among the 29 junction classes, seven junctions are subsumed by the remaining junctions, so we select the other 22 junction classes and generate 634 diverse routes. Based on the results shown in Table I, we can conclude that: (1) Even though CROUTE selects a much smaller set of junctions than CRT, the number of route types is the same. (2) CRT produces much more failed test cases due to the existing issues in Apollo or HD map, but the types of discovered issues are almost the same. It means that most of the failed test cases trigger the same issues. Hence, CROUTE can generate diverse test cases and reduce testing efforts. (3) Besides the six types of issues discovered by CROUTE, CRT discovered only one more bug: overshooting at a sharp turn, which means that Apollo fails to recognize the sharp turn on a junction lane and overshoots the lane boundaries. After further analysis, such sharp turns are due to map errors made by map makers. Hence, it is specific to locations on the map. No method can guarantee to discover such issues unless search for the whole map.

In conclusion, compared with the testing cost and issue discovery, the proposed junction classification in CROUTE is suitable and reasonable.

D. RQ3: Efficiency of CROUTE

To show the efficiency of CROUTE, we compare CROUTE with CRT and the random method. In CRT, we generate one test case for one route. There are 9026 feasible routes at the junctions on the map, resulting in 3696 base test cases, i.e., the combinations of incoming and outgoing road lanes connected directly by junction lanes. Thus, we need to generate at most 5330 mutated test cases, each of which covers a new route. While in CROUTE and the random method, test cases are mutated based on the two atomic mutation operators given in Fig. 3. In the random method, to guarantee the same number of selected junctions with CROUTE, we randomly select 22 junctions from the map and compute their failed test cases, covered route types, and discovered issues. We execute the random method 100 times and compute their averages. Fig. 5 shows the numbers of generated test cases, including the base test cases and the

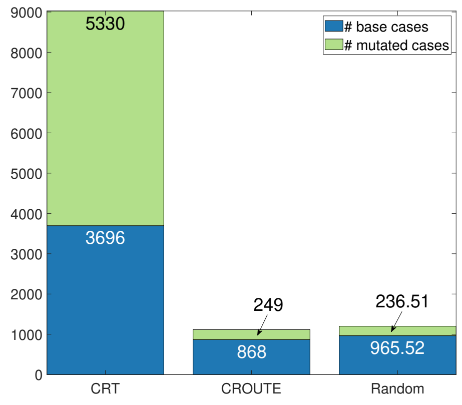


Fig. 5: The numbers of test cases generated by CRT, CROUTE, and Random.

mutated test cases, by each method, and Table I gives more details on the testing results.

From the results, we can find that compared with CRT, CROUTE can significantly reduce the number of test cases and guarantee the number of discovered issues. Even though the random method generates more test cases, the numbers of all route types and discovered issues are much less. It means that the random method produces more similar test cases, resulting in fewer discovered issues with a similar number of failed test cases. It also shows that CROUTE can generate diverse test cases with junction classification.

Due to insufficient re-routing, the ego vehicle always chooses to overtake and cannot re-route when there are obstacles ahead. Hence, all mutated test cases fail, not covering new routes. Based on Table I, during the execution of base test cases, CROUTE has 178 failed test cases, which trigger 5 types of issues, whereas CRT has 848 failed test cases but covers only one more variety of issues (i.e., overshooting at a sharp turn). Since the lane gap causes most failures, it has a higher probability of covering a new type if we test all routes with this type. The reason is that the routes in the same type may have a different lane width, so test cases running in the routes with smaller lane widths can be successful. Hence, CRT has a higher route coverage.

VI. CONCLUSION

In this paper, we propose a route coverage-based testing method, CROUTE, for AVs' route planning testing on a map. We first model the map as a labeled Petri net. Based on the Petri net, we classify junctions according to junction topology features and route features and propose a metric called route type to measure route diversity. A systematic test case generation method is proposed, which can generate test cases to cover all route types for a well-design ADS. The experiment results from testing on Apollo show the effectiveness and efficiency of CROUTE.

In the future, we will conduct more testing of other maps and other ADSs, such as Autoware. We will also investigate the improvement of the current method by considering lanes' shapes and widths. Moreover, we can also investigate global route planning testing on a map.

References

[1] Waymo, "Waymo," https://waymo.com/, 2016, online; accessed April 2020.

[2] Baidu, "Apollo open platform," http://apollo.auto/.

[3] Autoware.AI, "Autoware.AI," www.autoware.ai/.

[4] S. Feng, Y. Feng, C. Yu, Y. Zhang, and H. X. Liu, "Testing scenario library generation for connected and automated vehicles, part I: Methodology," *IEEE Trans. Intell. Transp. Syst.*, 2020.

[5] W. G. Najm, S. Toma, J. Brewer *et al.*, "Depiction of priority light-vehicle pre-crash scenarios for safety applications based on vehicle-to-vehicle communications," National Highway Traffic Safety Administration, U.S. Department of Transportation, Washington, DC, Tech. Rep. DOT HS 811 732, Apr. 2013.

[6] P. Nitsche, P. Thomas, R. Stuetz, and R. Welsh, "Pre-crash scenarios at road junctions: A clustering method for car crash data," *Accident Analysis & Prevention*, vol. 107, pp. 137–151, 2017.

[7] J.-P. Paardekooper, S. Montfort, J. Manders, J. Goos, E. d. Gelder, O. Camp, O. Bracquemond, and G. Thiolon, "Automatic identification of critical scenarios in a public dataset of 6000 km of public-road driving," in *Proc. Int. Technical Conf. Enhanced Safety of Vehicles*, 2019.

[8] W. Ding, M. Xu, and D. Zhao, "CMTS: A conditional multiple trajectory synthesizer for generating safety-critical driving scenarios," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020, pp. 4314–4321.

[9] S. K. Bashetty, H. B. Amor, and G. Fainekos, "Deepcrashtest: Turning dashcam videos into virtual crash tests for automated driving systems," in *Proc. IEEE Int. Conf. Robot. Autom.*, Paris, France, 2020, pp. 11 353–11 360.

[10] D. Zhao, H. Lam, H. Peng, S. Bao, D. J. LeBlanc, K. Nobukawa, and C. S. Pan, "Accelerated evaluation of automated vehicles safety in lane-change scenarios based on importance sampling techniques," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 3, pp. 595–607, 2017.

[11] M. Althoff and S. Lutz, "Automatic generation of safety-critical test scenarios for collision avoidance of road vehicles," in *Proc. IEEE Intell. Veh. Symp.*, 2018, pp. 1326–1333.

[12] M. Klischat and M. Althoff, "Generating critical test scenarios for automated vehicles with evolutionary algorithms," in *Proc. IEEE Intell. Veh. Symp.*, 2019, pp. 2352–2358.

[13] H. Beglerovic, M. Stolz, and M. Horn, "Testing of autonomous vehicles using surrogate models and stochastic optimization," in *Proc. IEEE Int. Conf. Intell. Transp. Syst.*, 2017, pp. 1–6.

[14] S. Feng, Y. Feng, H. Sun, S. Bao, Y. Zhang, and H. X. Liu, "Testing scenario library generation for connected and automated vehicles, part II: Case studies," *IEEE Trans. Intell. Transp. Syst.*, 2020.

[15] R. Chen, R. Sherony, and H. C. Gabler, "Comparison of time to collision and enhanced time to collision at brake application during normal driving," SAE International, Tech. Rep. 2016-01-1448, 2016.

[16] G. E. Mullins, P. G. Stankiewicz, and S. K. Gupta, "Automated generation of diverse and challenging scenarios for test and evaluation of autonomous vehicles," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 1443–1450.

[17] G. E. Mullins, A. G. Dress, P. G. Stankiewicz, J. D. Appler, and S. K. Gupta, "Accelerated testing and evaluation of autonomous vehicles via imitation learning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 1–7.

[18] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 4, pp. 1135–1145, Apr. 2016.

[19] J. Bohren, T. Foote, J. Keller, A. Kushleyev, D. Lee, A. Stewart, P. Vernaza, J. Derenick, J. Spletzer, and B. Satterfield, "Little Ben: The Ben Franklin Racing team's entry in the 2007 DARPA Urban Challenge," *J. Field Robot.*, vol. 25, no. 9, pp. 598–614, 2008.

[20] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *Int. J. Robot. Res.*, vol. 28, no. 8, pp. 933–945, 2009.

[21] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 1478–1483.

[22] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a Frenét Frame," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2010, pp. 987–993.

[23] W. Xu, J. Wei, J. M. Dolan, H. Zhao, and H. Zha, "A real-time motion planner with trajectory optimization for autonomous vehicles," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2012, pp. 2061–2067.

[24] Y. Zhou, H. Hu, Y. Liu, S.-W. Lin, and Z. Ding, "A real-time and fully distributed approach to motion planning for multirobot systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 49, no. 12, pp. 2636–2650, 2017.

[25] C. Liu, S. Lee, S. Varnhagen, and H. E. Tseng, "Path planning for autonomous vehicles using model predictive control," in *Proc. IEEE Intell. Veh. Symp.*, Jul 2017, pp. 174–179.

[26] J. Wei, J. M. Snider, T. Gu, J. M. Dolan, and B. Litkouhi, "A behavioral planning framework for autonomous driving," in *Proc. IEEE Intell. Veh. Symp.*, Jun. 2014, pp. 458–464.

[27] S. Aoki and R. Rajkumar, "Dynamic intersections and self-driving vehicles," in *Proc. ACM/IEEE Int Conf Cyber-Physical Syst.*, 2018, pp. 320–330.

[28] K. Chellapilla, "Rethinking maps for self-driving," https://medium.com/lyftlevel5/https-medium-com-lyftlevel5-rethinking-maps-for-self-driving-a147c24758d6, Oct. 2018.

[29] LG Electronics, "Apollo 5.0 (forked)," https://github.com/lgsvl/apollo-5.0/tree/1b7a466ada5aa1d50a5684bd8885ccee11a7187, 2019.

[30] ——, "LGSVL simulator," https://github.com/lgsvl/simulator/releases/tag/2020.01, 2019.

[31] ——, "Map for San Francisco," https://content.lgsvlsimulator.com/maps/sanfrancisco/, 2020.