Simulation-based Autonomous Vehicle Verification: An Exploration of Agency-based Test Generation Methods

Greg Chance[1,3], Abanoub Ghobrial[1,3], Severin Lemaignan[2,3], Tony Pipe[2,3], Kerstin Eder[1,3]

***Abstract*—Simulation-based verification is beneficial for assessing otherwise dangerous or costly on-road testing of autonomous vehicles (AV). This paper addresses the challenge of efficiently generating effective tests for simulation-based AV verification using software testing agents. The multi-agent system (MAS) programming paradigm offers rational agency, causality and strategic planning between multiple agents. We exploit these aspects for test generation, focusing in particular on the generation of tests that trigger the precondition of an assertion. On the example of a key assertion we show that, by encoding a variety of different behaviours respondent to the agent's perceptions of the test environment, the agency-directed approach generates twice as many effective tests than pseudo-random test generation, while being both efficient and robust. Moreover, agents can be encoded to behave naturally without compromising the effectiveness of test generation. Our results suggest that generating tests using agency-directed testing significantly improves upon random and simultaneously provides more realistic driving scenarios.**

***Index Terms*—Test Generation, Simulation, Autonomous Driving, Verification, Multi-Agent System, Test Agent**

I. Introduction

VERIFICATION is the process used to gain confidence in the correctness of a system with respect to its requirements [7]. Testing is a technique that can be used to achieve this by showing that the intended and actual behaviours of a system do not differ and detecting failures against the requirements in the process [37].

Using simulation to test autonomous driving functions in safety critical scenarios benefits from full control over the environment, where road layouts, weather conditions, a variety of road users and other driving scenario parameters can be directed to achieve specific test targets. These tests may aim to provide evidence to regulators of the functional safety of the vehicle or its compliance with commonly agreed upon road conduct, such as the Vienna convention [38], typically implemented at national level as a set of rules [32], road traffic laws and penalties [35].

Verification of complex systems is challenging. In semiconductor design, for example, it has long been recognised that verification can take up to 70% of the design effort [3], with the largest part still being achieved with simulation-based techniques. The testbench is the code used to drive a stimulus sequence into the Design under Verification (DUV) while observing input protocols. It also records coverage and checks the DUV's response. The testbench provides a completely closed environment from the DUV's perspective. Simulators are used to execute testbenches. Automation plays a critical role in achieving verification targets efficiently and effectively.

Coverage-driven verification is a systematic, goal-directed simulation-based verification method [2] that offers a high degree of automation and is capable of exploring systems of realistic detail under a broad range of environment conditions. Because exhaustive simulation is intractabe due to the vast parameter space, the remaining challenge is in strategically selecting the (ideally smallest set of) test cases that result in the highest level of confidence in the design's correctness. Automating test generation has been the focus of research for decades, giving rise to a variety of coverage-directed stimulus generation techniques that exploit formal methods, genetic programming and machine learning [19].

Compared to semiconductor design verification, AV verification faces even bigger challenges, including automatic test generation. It is well known that few of the valid tests are actually interesting from a verification point of view. Estimates vary, but demonstrating AV safety with a confidence of 95% that the failure rate is at most 1.09 fatalities per 100 million miles driven would take 275 million miles, equivalent to 12.5 years for a fleet of 100 AVs [20]. This figure is based on the number of roads fatalities in the US, and would need to be adjusted taking into consideration local statistics on road safety, e.g. the number of road fatalities per billion vehicle-km in the UK has been half that of the US in 2018 [29]. Ways must be found to test the scenarios of interest without needing millions of miles of driving or billions of miles of simulated driving [24]. In particular, simulation-based testing offers the opportunity to increase the number of otherwise rare events [23] in order to determine whether the AV handles such rare events appropriately.

Considering the AV as a DUV, the challenge is to generate tests that interact with the AV over a period of time, thereby creating an environment in which the AV needs to respond to the received stimulus while making progress towards its destination. As such, the AV can be classed as a *responder* DUV, i.e. a DUV that reacts to lower-level stimulus observed on its interfaces with the surrounding environment in order to maintain legally correct driving behaviour and follow the

[1]Greg Chance, Abanoub Ghobrial and Kerstin Eder are with the Trustworthy Systems Lab, University of Bristol, Bristol, UK {greg.chance, abanoub.ghobrial, kerstin.eder}@bristol.ac.uk

[2]Severin Lemaignan and Tony Pipe are with the University of the West of England, Bristol, UK {severin.lemaignan, tony.pipe}@brl.ac.uk

[3] Bristol Robotics Laboratory, University of the West of England, Bristol, UK

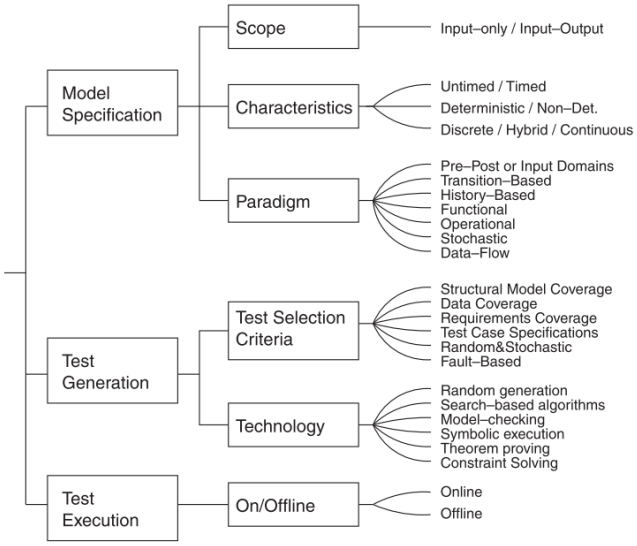

Fig. 1. Taxonomy of model-based test generation, from [37].

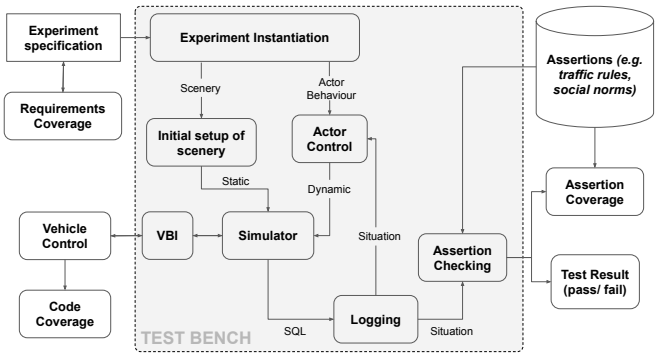

Fig. 2. Testbench Architecture.

social norms associated with road traffic.

This paper investigates the benefits of introducing *agency* into the verification environment in order to address the challenges of verifying the responder DUV. Each software agent is tasked with specific goals that aim to achieve verification objectives, e.g. reaching coverage targets. A set of software agents can then be directed to interact, coordinating their behaviour in response to the AV's observed actions in order to increase the likelihood of rare events occurring during simulation to reach coverage targets faster.

Our key research question is: what are the benefits of using agent-based test generation for the verification of AVs in simulation? In particular, we are interested in how agency-directed test generation compares to pseudo-random test generation techniques wrt. the following criteria for a 'good' test case, which are inspired by [14]:

effectiveness in detecting failures, *efficiency* in minimising the number if tests required to achieve verification goals, *economy* in terms of resource usage and also *robustness* towards changes. Our results suggest that generating tests using directed agents significantly improves upon random and simultaneously provides driving scenarios that are more realistic than those obtained by random test generation.

We regard agent-based test generation as a contribution to the well-established model-based test generation paradigm. A taxonomy of model-based test generation from [37] is given in Figure 1. The agent-based technique creates two new entries in that taxonomy, a new Paradigm under Model Specification, *Agent-based*, and a new Technology under Test Generation, *Agency*, which includes reactive reasoning, causality and strategic planning between multiple agents. In this paper we use an agent-based model to specify the test environment of the AV. Agency is given to the key dynamic entities in the test environment that interact with the AV, in our case pedestrians. Agency, implemented through a belief-desire-intention agent interpreter such as Jason [9] is then employed to generate tests based on the multi-agent system that represents the test environment. Note that other test generation techniques, such as random generation, which we use as baseline for evaluation, or model checking [8], can also be applied to an agent-based model.

This paper is structured as follows. In the next section we introduce the terminology we will adopt throughout the paper and present the testbench architecture used in our experiment. In Section III we review related work on test generation for simulation-based AV verification and introduce the basics of multi-agent systems. Section IV presents our case study, which is centred around test generation for a collision avoidance scenario. Results are presented in Section V and evaluated in Section VI. We conclude in Section VII and give an outlook on future work.

II. Terminology and Testbench Architecture

We will adopt the terminology defined in [36], where *scene* refers to all static objects including the road network, street furniture, environment conditions and a snapshot of any dynamic elements. Dynamic elements are the elements in a scene whose actions or behaviour may change over time, these are considered actors and may include other road vehicles, cyclists, pedestrians and traffic signals. The *scenario* is then defined as a temporal development between several scenes which may be specified by specific goals and values. A *situation* is defined as the subjective conditions and determinants for behaviour at a particular point in time.

The proposed testbench, see Fig. 2, is driven by a specification for the experiment which defines the scene and scenario including all dynamic actors. The experiment or test case specification specifies the test inputs, i.e. the execution conditions for an item to be tested [18]. The vehicle behaviour interface (VBI) connects the AV controller (vehicle control) to the simulator. It provides the simulator with the driving decisions of the AV and forwards updates on the scene to the AV controller. A geospacial database logs the AV and all other actors to enable post-simulation assertion checking.

Given the experiment specification, tests may be generated in a variety of ways that differ mainly in their effectiveness and efficiency. Manually generated tests are typically effective in achieving verification objectives, but are considered

expensive due to the high engineering cost involved. Random methods are usually employed at the early stages of testing to build up coverage quickly. They suffer, however, from a high number of invalid tests being generated and, even when constrained to produce only valid tests, the tests generated are often not interesting wrt. the verification objectives, in our case this refers to exercising the collision avoidance decision making logic of the AV. Model-based test generation offers an alternative that produces valid and interesting tests at the cost of developing a model that faithfully encodes the behaviour of the test environment. This model can then be explored in a variety of ways, see Figure 1.

Our paper explores how the agency that is naturally present in the multi-agent system that represents the dynamic actors in a scene can be used for model-based generation of test scenarios in the context of simulation-based AV verification.

One could argue that the simulation environment presented to the AV should be as close to the real world it seeks to emulate as the most realistic proving ground. Such efforts seek to reduce the *reality gap*, providing the most likely scenarios and actor behaviour. This can be achieved by monitoring real traffic scenarios, e.g. tracking individual vehicles, cyclists and pedestrians, and building behavioural models for each of the tracked entities [6]. On the other hand, the creation of edge cases is critical to reach verification objectives in a timely manner. Edge cases are events that occur rarely under normal circumstances, yet realistic *and* critical for gaining confidence in the correct behaviour of the AV. This is exactly what the agent-based test generation approach introduced in this paper is aimed at.

III. Background

A. Related Work

An overview of the challenges currently faced in software testing of AVs is given by Koopman, highlighting the importance of fault injection into the testing domain [22] and how to decide what aspects of the system need to be tested in the areas of operational design domains, event detection and vehicle manoeuvres [21]. Describing a driving scene in natural language has been shown to be an effective framework for scenario generation. This can also be automated based on formalised rules and knowledge [4] or even evolved from ISO safety standards [27]. Hallenbach et al. take the approach to test generation of developing composite metrics for traffic quality and using them to determine if a scenario is 'critical' or not and therefore worth exploring further [17]. Generating targeted test cases can be a more efficient way to achieve verification objectives. For example, Mullins et al. [28] describe a test generation method that is focused on exploring the transitions between decision making performance modes of the AV, with the aim to find tests that exercise the complete decision making logic. Saigol et al. [31] discuss the need for *smart actor control* which they expect may lead to interesting scenarios when these interact with the AV a testbench framework for automated testing. Rocklage et al. [30] approach the problem of test generation from the viewpoint of the other traffic participants, using a trajectory planner to ensure coverage over a fixed list of scene parameters. Tuncali et al., [33] use rapidly exploring random trees to explore boundary scenarios for different adversarial road users. Exploiting agency for test generation, however, has not yet been investigated.

B. Multi-Agent Systems

Georgeff and Lansky were key in the development of the belief-desire intention (BDI) agent programming approach [16] and also in the early work of multi-agent systems [15]. In a multi-agent system, multiple intelligent agents interact in order to collectively solve problems that are too difficult or impossible for individual agents to achieve. A multi-agent system includes a set of software agents and their environment. Agents can be equipped with varying degrees of agency, starting from passive agents, such as obstacles, including active agents that have goals of low complexity, such as pedestrians that act as part of a group, to rational agents that are capable of reasoning based on a cognitive model of the situation, their perception of the surrounding environment, and a set of rules they can use for strategic planning and communication with other agents in the system. The individual agents are considered *autonomous* and in control of their behaviour within the multi-agent system as they pursue their goals. This results in self-organisation and self-direction towards achieving a common goal.

Combining the BDI framework with an automated test generation approach leads to the concept of a software agent capable of generating tests. Such intelligent, agent-based test generation has first been applied in the human-robot interaction domain [1], where a coverage-driven test generation approach was supplemented with reinforcement learning to improve the efficiency and effectiveness of testing the critical part of a robot-to-human handover task within a collaborative manufacturing scenario. Test agents have also been proposed by Enoiu et al. [12] for regression testing, although they are used more for test selection from a library of tests, rather than for test generation. These agents use inter-agent messaging to decide what tests to execute and with what prioritisation.

C. Reinforcement Learning Agents

Background on reinforcement learning. Any background on RL used for test generation?

IV. Case Study

This section describes a case study to explore the proposed agent-based test generation method.[1] Our aim is to generate tests that exercise an assertion that requires the AV to avoid collisions with other road users, provided that such road user, a pedestrian in this case, intrudes in the path of the AV in such a way that a collision can be avoided by the AV either by braking or manoeuvring. This is similar to standardised testing, e.g. Euro-NCAP CPNA-25 [13]. In Fig. 3 this is illustrated by the *precondition zone*, i.e. the area of interest in which the assertion is activated.

[1]The code used is available at github.com/TSL-UOB/CAV-MAS.

In the following, we shall term tests that activate the assertion *successful tests*; these count towards assertion coverage. Intrusions that fall within the *stopping distance* of the AV [32], i.e. the 12m long zone directly ahead of the AV when driving at 30km/h, equivalent to ∼20mph (miles per hour) as marked in Fig. 3, are not considered interesting as they result in unavoidable collisions; tests of this type are of limited use [34]. Thus, in this case study, unavoidable collisions are those with a time to collision (TTC) of less than 1.33s. The extent of the *precondition zone* is limited to a single simulation tick (1.0s) which puts the upper limit at 2.33s ahead of the *stopping distance*.

A single assertion is chosen so that we can study the fundamental properties of agent-based test generation in a simple setting in comparison to using random test generation techniques. Moreover, in this investigation we do not consider the test result (pass or fail), i.e. the behaviour of the AV in response to the assertion being activated.

A. Test Environment A: Straight Road

The test environment, depicted in Fig. 3, is a straight two-lane 99m road with two 6m wide lanes. Pavements are on both sides of the road which are three meters in width giving a total area of 18m×99m. The AV velocity is 9.0m/s (equivalent to ∼20mph) and the pedestrian velocity is 1.4 m/s (equivalent to ∼3mph). The map is discretised with 1.5m resolution for simple division into the AV and pedestrians' velocities. Thus, in the discretised world the AV velocity is 6 cells/s and the pedestrians' velocity is roughly 1 cell/s. The total number of map cells is 12×66 = 792 cells.

The AV travels along the left-hand lane of the road starting at cell $y = 0$ and travelling to the end, cell $y = 66$. If the precondition satisfied or the AV reaches the end of the road, then the test is restarted. The AV occupies an area 4.5m×3m equivalent to 3×2 cells. There are no other vehicles and the right-hand side of the road is unoccupied.

To activate the assertion, the precondition must be satisfied, i.e. an agent has entered the *precondition zone* of the AV, which extends 6 cells forwards of the *stopping distance* of the vehicle as shown in Fig. 3. Under the given conditions, it is impossible for an agent to move into the AV's path from some pavement areas within the test environment. These are marked in dark grey in Fig. 3 and labelled invalid spawn locations.

The environment is initialised with the agents spawned randomly on any valid spawn location within the pavement area. When the environment is reset, new random locations are chosen for the agents. Random locations are controlled by using a fixed seed that is based on the experiment number; thus, initial spawn locations become repeatable (through using the same seed), which ensures a fair comparison between different experiments.

B. Test Agents

The test agents in our case study are pedestrians. At the start of a test, pedestrians are randomly located onto the pavement area using valid spawn locations. The AV is a vehicle that drives at constant speed in a straight line, it does not brake or

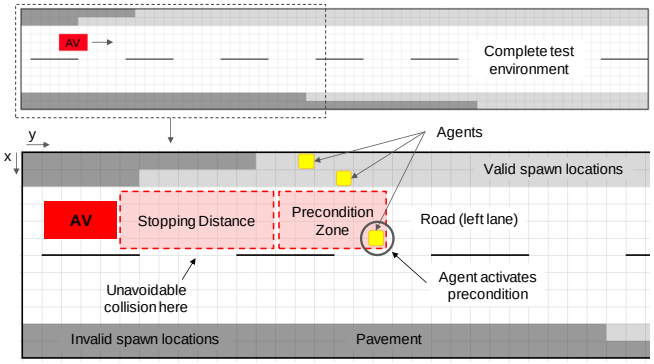

Fig. 3. Test environment at full scale (upper part) and in detail (lower part), marking valid and invalid spawn locations for the pedestrians and indicating the position and direction of the AV (upper part).

turn. For the purpose of comparing the performance of agent-based test generation techniques, the test agents have different behaviours from simple random (i.e. no direction through agency) to more directed, strategic planning-based agency, details of these are provided in this section. We distinguish three main classes of behaviours, random, agency-directed and Q-learning, see also Table I.

The number of agents generated per test should also be considered; too few and the likelihood of an agent activating the assertion will be low and heavily dependent on the initial starting position of the agent in the test environment, the size of the environment and differences in speed. The number of agents is physically limited by the number of available grid locations (1.5m spacing) on the pavement. As the number of agents, nA, increases, it is expected that the probability of activating the assertion using any behaviour increases significantly as more agents appear in the road, with a corresponding increase in computational expense. In comparison, when using more intelligent, agency-directed behaviour, we expect that fewer agents can activate the assertion precondition more readily, thus resulting in shorter, less complex and hence more efficient tests. The range of nA explored in this case study is from 1 to 20.

1) Random Based: For the random class, *random* behaviour means that the test agent can perform any random action at each simulation tick, with the available actions being: do nothing (stand still), move forward, backward, left or right. In *constrained random* mode, the pedestrian is initially walking along the pavement and has only one action available; to randomly choose when to cross the road. The constrained random behaviour is included so that a comparison between crossing the road at a targeted or at a random time can be made.

Algorithm 1 Random Action

1: **while** $t < t\max$ **do**
2: $a = random.randint(1, 5)$
3: **end while**

Algorithm 2 Constrained Random

```
while t < tmax do
    ε_r = random.randint(1, 9)
    if ε_r == 0 then
        a = cross_road
    else
        a = walk_pavement
    end if
end while
```

2) Reactive Agency: The second class of behaviours is agency-directed, taking into account the agent's perceptions for strategic planning. Agents are initially walking along the pavement as in constrained random mode. The *proximity* behaviour instructs the agent to cross the road when the AV is within a certain radius.

Algorithm 3 Proximity

```
while t < tmax do
    P = distance.cityblock(AV, Agent)
    if proximity <= 15 then
        a = cross_road
    else
        a = walk_pavement
    end if
end while
```

The *election* behaviour ranks agents within a proximity radius to the AV and elects the agent with the shortest distance to the AV. The trigger radius is 15 cells using city-block measure and is the same value for both *proximity* and *election* behaviours. The major difference between the agent-directed behaviours is that the *election* behaviour will only elect a single agent to cross the road whereas the *proximity* behaviour will result any number of agents crossing the road as long as they are within range. While the *proximity* behaviour may lead to the desired coverage it is less realistic than the *election* behaviour.

Algorithm 4 Election

```
while t < tmax do
    for A = 0 : nA do
        P(A) = distance.cityblock(AV, A)
    end for
    if min(P <= 15) then elected_agent = min(P)
    end if
    if elected_agent then
        a = cross_road
    else
        a = walk_pavement
    end if
end while
```

3) Reinforcement Learning: Q-learning can be used to identify an optimal action-selection policy without an explicit environment model in controlled Markovian domains through agents experiencing the consequences of their actions [39]. Actions (a) are selected using temporal difference method of expected rewards R which are discounted by γ. For a given state s at time t, the Q value is given by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot \left[r_t + \gamma \cdot \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{1}$$

where α is the learning rate and Q' is the updated q value given the agent took action a in state s at time t and gets reward r plus the future discounted value of $\max_a(Q)$ at time $t+1$ where the addition refers to another episode. The major limitation of this method is that it requires lookup tables to describe the Q values for the state-action space which may be computationally cumbersome for large domains not to mention the time for the agents to explore each state and action combination. A variation of Q-learning to overcome the infinite state-space problem using function approximation [5] can be adapted here for the use of agents in open and dynamic environments. Using a fixed set of independent linear functions, the Q-learning function can be described as a weighted sum of the functions:

$$Q(s_t, a_t) = w_1\theta_1(s, a) + ... + w_i\theta_i(s, a) \tag{2}$$

where w is the weight associated with each functional approximation of the state-space $\theta(s, a)$. After each episode the current reward and discounted future value represents the temporal difference learning element, δ:

$$\delta = r_t + \gamma \cdot \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{3}$$

and the weight update if performed based on the temporal difference element and the learning rate (α):

$$w_i \leftarrow w_i + \alpha \cdot \delta\theta_i(s, a) \tag{4}$$

The algorithm for the *q-agent* is shown in Algorithm 5 where $action\_space$ contains the valid actions for each agent given a limited (non-infinite) domain.

Algorithm 5 Q-Agent

```
while t < tmax do
    Sample s_t
    Get Q(s_t, a_t) using (2)
    Sample s_{t+1} from action_space
    Update Q(s_t, a_t) using (1)
    Select a from max_a(Q(s_t, a_t))
    Update ego and agent(s) state
    Get r_t
    Update w_i using (4)
end while
```

Convergence proof of the Q-learning theorem has been given [25] although this is assuming infinite exploration in a controlled Markovian domain, discretised and deterministic [39]. Q-learning using linear function approximation has also been proven by Melo et al. [26]. Implementing Q-learning with a vehicle controller may not be considered deterministic

due to the use of neural networks in the perception stack[2] so may fail many of these convergence assumptions. However, when using these algorithms in a simulation domain, control over the input space can be regulated to give repeatable results under certain conditions [10].

C. Scoring

In an attempt to encourage more realistic agent behaviour, a basic scoring system is used to direct agent behaviour by penalising or rewarding certain actions. In particular, a living cost of one is charged for each elapsed time step to promote shorter tests, and a penalty of five is issued for each time step that an agent spends in the road. This is based on the general observation that most pedestrians do not predominately walk in the road but rather cross over it and hence higher scores are associated with less time spent in the road. A reward of 100 is given if the agent enters the AV's *precondition zone* More sophisticated scoring systems will be explored in our future work.

D. Simulation and Logging

Each agent behaviour was implemented in Jason [9] and executed within the testbench environment as described in Section II. During simulation, the agents' actions, scores and time to test completion were recorded in a log.

Each agent behaviour was repeated 1000 times and the number of successful tests was counted. A successful test is one that has a pedestrian intrude into the *precondition zone* of the AV, thereby creating the opportunity for a subsequent intersection with the AV, i.e. a collision, thus triggering the AV's collision avoidance decision making logic.

A list of random start locations was created for the pedestrians for all settings of agent numbers, nA. This list was used to spawn the pedestrians for each of the agent behaviours to ensure the initial conditions between experiments were identical.

V. Results

The results are evaluated based on five criteria; Accuracy is the ratio of successful tests the agents generated to the number of all tests, Score is a measure of how natural the agents behaved, Combined Score combines accuracy with score, Agent CPU Time is a measure of computational efficiency, and Test Generation Time is how long the agents took to generate the test in both simulation ticks and wall clock time. Both Score and Test Generation Time have distributions associated with them and therefore confidence intervals are provided.

A. Test Accuracy

Test accuracy, defined as the number of tests that have activated the precondition for the assertion as a ratio of all tests, is shown for each agent behaviour in Fig. 4. The *random* and *constrained random* behaviours have significantly lower accuracy compared to the directed agent behaviours when $nA < 10$. For high nA the accuracies converge mostly due to a saturation of agents in the test environment. Fig. 4 also shows that for $nA = 1$ a directed agent outperforms a random agent by more than 2:1. The *election* behaviour generates a slightly higher accuracy than the *proximity* agent behaviour, but only by 2.2%. However, for $nA = 2$ and above the *proximity* agent behaviour shows a 10% increase in accuracy over the *election* behaviour; this is because agents with this behaviour have multiple attempts to trigger the precondition whereas the election behaviour permits only one.

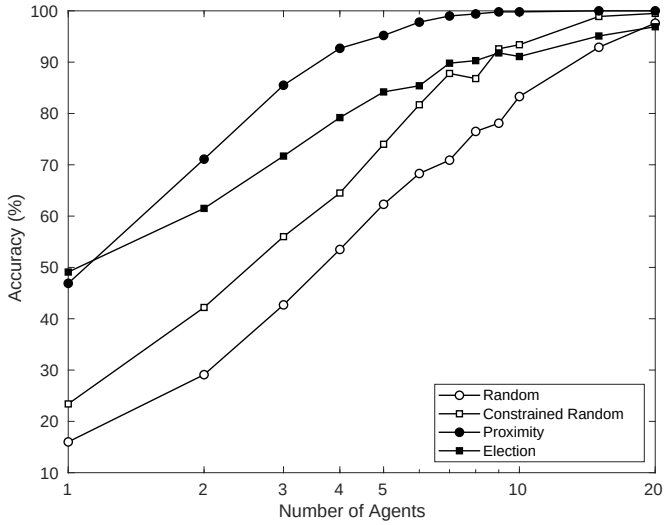

Fig. 4. Accuracy of agent behaviours to generate successful tests.

B. Test Score

For tests that activate the precondition the average agent score is compiled including 95% confidence intervals, see Fig. 5. The maximum theoretical score of any agent is 94, which includes 100 points for a successful test subtracting a living cost of one and a road penalty of five; this requires the agent to spawn adjacent to the *precondition zone*. For a single agent, scores are similar for all agent behaviours, as only successful tests are included in the scoring. As the number of agents increases, the random behaviours diverge from the directed behaviours, and the variance observed for the random behaviour scores increases. This indicates that the directed agents display more natural behaviour than those using random.

As nA increases, the random agents are more often found in the road and hence their average score decreases. In contrast, the directed agents are only found crossing the road when they deem fit, and they remain on the pavement at all other times, keeping the score much higher and resulting in more realistic test cases. The basis for this rationale is from the general observation that people walk on pavements rather than roads and hence walking in the road is penalised. No significant difference between the two directed behaviours can be observed in the score, even with a high number of agents. This indicates that both result in similar level of realism

[2]neural networks with fixed weights and no adaptive learning are deterministic mathematical structures within the data used to train them but in a practical application there is no such control over the input data space

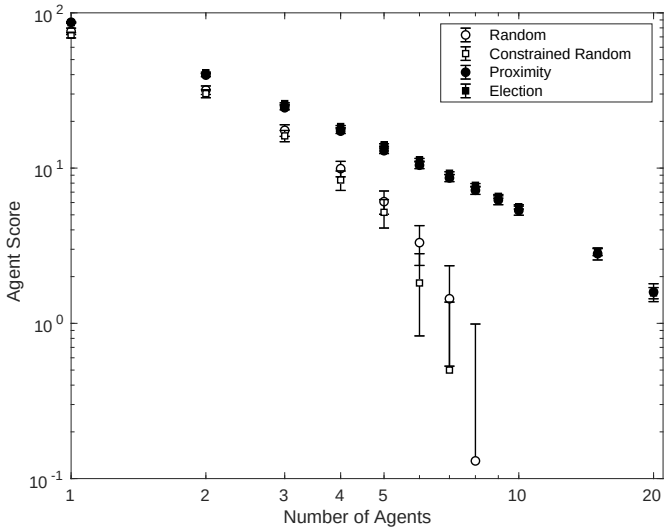

Fig. 5. The average agent score for successful tests with 95% confidence intervals for each agent behaviour.

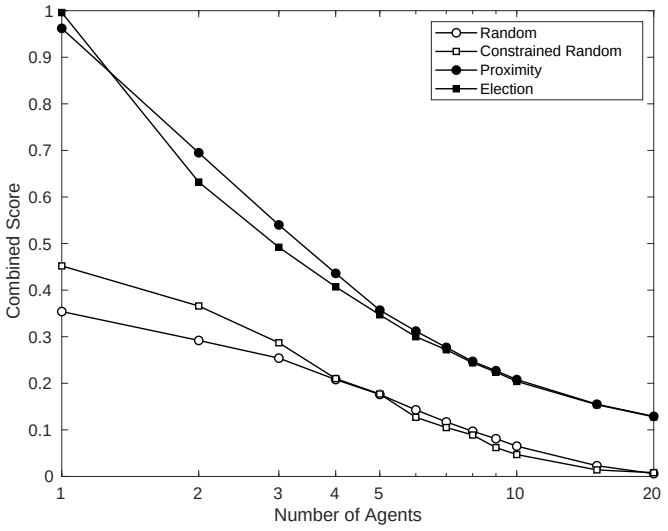

Fig. 6. The combined score including accuracy and the original score for each agent behaviour.

in agent behaviour, at least within the limited scope of this example.

C. Combined Score

As the score in Section V-B only includes tests that meet the precondition, it is tempting to interpret these results as overly optimistic. To counteract this we use a *combined score* that is calculated based on the score and accuracy using (score * accuracy / 1000). This combined measure promotes scores that are attached to high accuracies, describing agents that can generate successful tests with natural pedestrian behaviour. The normalised results, Fig. 6, show that the directed agents are more than twice as effective within this new combined definition than agents with random behaviour for $nA = 1$, although this lead drops rapidly with increasing agent numbers, reaching a steady performance gap of around 12%. The *constrained random* agent behaviour outperforms the *random* for $nA < 4$ beyond which there is little difference. Therefore, if random behaviour is chosen, then the extra effort in implementing constrained random may not be worthwhile considering the low returns, except when using low agent numbers. The *election* agent behaviour has the highest combined score for $nA = 1$, but as agent numbers increase, better results can be seen for the *proximity* agent behaviour up to $nA = 4$, beyond which there is no significant difference between the two directed agent behaviours.

D. Agent CPU Time

For each successful test, the CPU time used to execute the agent actions was averaged over the 1000 runs and assessed across different agent behaviours and numbers of agents, Fig. 7. This allows comparing the resources required to execute agents of different behaviour types. The more complex agents have additional CPU overhead compared to agents with random behaviour. This difference remains relatively stable as agent numbers increase.

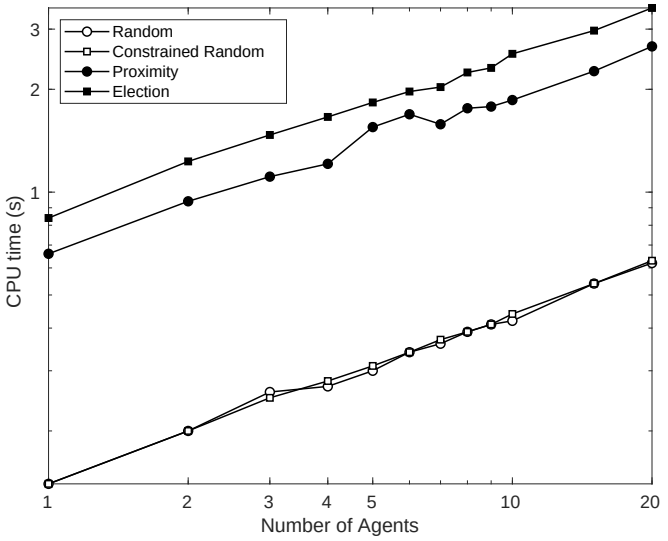

Fig. 7. The CPU time, t_C, to execute agent actions averaged over 1000 runs.

E. Test Generation Time

For each successful test generated, the number of simulation ticks consumed during test generation was averaged over 1000 runs and compared across different agent numbers for all four agent behaviours, Fig. 8. The directed agents show improved performance over the agents with random behaviour for $nA = 1$ by approximately a single simulation tick and this trend continues as agent numbers increase. By $nA = 20$ the directed agents are 1.85 simulation ticks faster than the agents with random behaviour. Overall, the *proximity* agents find tests in the smallest number of simulation steps.

VI. Evaluation

The results are assessed against the four criteria identified in Section I.

Effectiveness and Efficiency: How effective is the method at generating tests that detect failures in the DUV, and how

TABLE I
TEST AGENT SUMMARY TABLE SHOWING DESCRIPTION AND NUMBER OF LINES OF CODE (LOC) FOR EACH AGENT AND SAMPLE OF RESULTS: ACCURACY, COMBINED SCORE (s_c), CPU TIME (t_c) AND TEST GENERATION TIME (t_g) FOR $nA = 3$.

Behaviour	**Agent Description**	LOC	Accuracy (%)	s_c (points)	t_c (s)	t_g (ticks)
Random	Randomly perform action (forward, backward, left, right, stop)	23	42.7	0.254	0.09	9.11
Constrained Random	Walk along pavement, randomly cross the road	82	56.0	0.287	0.08	8.83
Proximity	Walk along pavement, cross road when AV in range	86	85.5	0.540	0.37	6.79
Election	As in Proximity but elect a single agent to cross	235	71.7	1.470	0.49	6.59

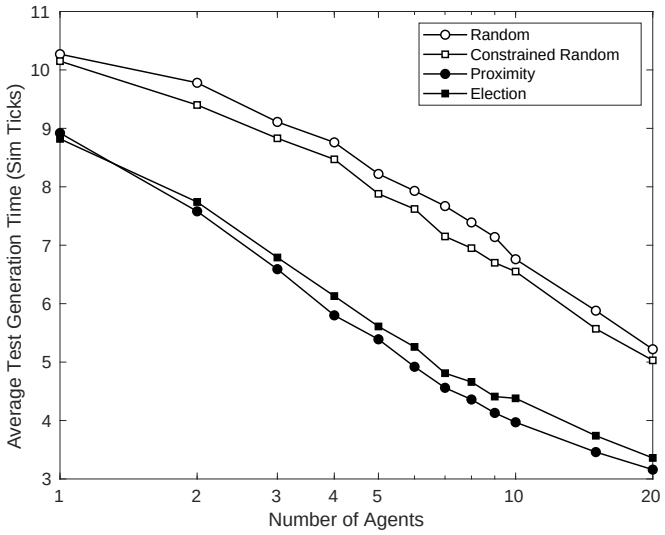

Fig. 8. The average time taken for an agent to find a successful test, t_g.

efficient is the method in minimising the number of tests required to achieve verification goals (i.e. assertion coverage)? The accuracy metric, defined above, shows how often an agent generates a test that satisfies the precondition of the assertion, thereby triggering the execution of the AV's collision avoidance decision-making logic, creating the potential to reveal defects in the DUV and achieving assertion coverage in the process. Fig. 4 shows that a small number of directed agents is around twice as effective as random ones and over three times as effective as a single agent.

Economy: How costly is the test case to develop and run? The resource cost (CPU time) to execute the agent actions, Fig. 7, is 4-5 times higher for the directed agents than for agents with random behaviour, see also heading t_c in Table I for $nA = 3$. However, the test generation time (t_g), Fig. 8, indicates that on average the directed agents find successful tests faster than random. Therefore, although more resource needs to be allocated to the directed agents, overall they find test cases faster due to higher efficiency. Wrt. development effort, the lines of code for each agent behaviour, see Table I, show that for a moderate investment the *random* behaviour can be improved significantly to obtain the performance of the *proximity* behaviour. However, diminishing returns are evident beyond that. This simple scenario would suggest that the level of agent complexity should be considered carefully as a simpler level of agency could potentially be more beneficial than more complex options.

Robustness refers to the maintenance required to adapt tests to software changes, i.e. different *scenes*. In the case study each test generated is of a different scene as the agent positions are randomly generated. The directed agents show higher robustness based on their accuracy when compared to random behaviours.

Overall, our results show that agency-directed testing outperforms random based, confirming that even a small amount of agency can be a distinct advantage over random techniques.

VII. CONCLUSION AND FUTURE WORK

The MAS programming paradigm offers rational agency, causality and strategic planning to software agents; these have been exploited in this research for test generation. On the example of a key assertion on collision avoidance we show that, by encoding a variety of different behaviours respondent to the agent's perceptions of the test environment, the agency-directed approach generates twice as many effective tests than a pseudo-random approach. Furthermore, agents can be encoded to behave more realistically without compromising their effectiveness. Our results suggest that generating tests using testing agents is a promising avenue of research and has been shown to significantly improves upon random whilst simultaneously providing more realistic driving scenarios.

Future work will extend this initial study to multiple assertions and a significantly larger variety of scenes including different road networks. As discussed in [11], agents could have their goal selection modified based on coverage feedback, giving rise to agent-based coverage-directed test generation. Abstracting agent perceptions to a feature based representation could ensure the agent state space scales up to large physical maps and is adaptable to new features as more assertions are added. Including personality in agents is also another avenue that could provide a tuning parameter [40] to generate edge cases.

ACKNOWLEDGEMENT

This research has in part been funded by the ROBOPILOT and CAPRI projects. Both projects are part-funded by the Centre for Connected and Autonomous Vehicles (CCAV), delivered in partnership with Innovate UK under grant numbers 103703 (CAPRI) and 103288 (ROBOPILOT), respectively.

REFERENCES

[1] D. Araiza-Illan, A. G. Pipe, and K. Eder. "Intelligent Agent-Based Stimulation for Testing Robotic Software in Human-Robot Interactions". In: *Proceedings of the 3rd Workshop on Model-Driven Robot Software Engineering*. MORSE'16. ACM, 2016, pp. 9–16.

[2] D. Araiza-Illan, D. Western, A. Pipe, and K. Eder. "Coverage-Driven Verification — An Approach to Verify Code for Robots that Directly Interact with Humans". In: *Hardware and Software: Verification and Testing (HVC'15)*. Springer, 2015, pp. 69–84.

[3] W. M. Arden. "The international technology roadmap for semiconductors perspectives and challenges for the next 15 years". In: *Current Opinion in Solid State and Materials Science* 6.5 (2002), pp. 371–377.

[4] G. Bagschik, T. Menzel, and M. Maurer. "Ontology based Scene Creation for the Development of Automated Vehicles". In: *IEEE Intelligent Vehicles Symposium*. Vol. 2018-June. 2018, pp. 1813–1820.

[5] L. Baird. "Residual algorithms: Reinforcement learning with function approximation". In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 30–37.

[6] F. Behbahani et al. "Learning from demonstration in the wild". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 775–781.

[7] J. Bergeron. *Writing testbenches — Functional verification of HDL models*. Springer, 2012.

[8] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. "Verifying Multi-agent Programs by Model Checking". In: *Autonomous Agents and Multi-Agent Systems* 12.2 (Mar. 2006), pp. 239–256.

[9] R. H. Bordini, J. F. Hübner, and R. Vieira. "Jason and the Golden Fleece of agent-oriented programming". In: *Multi-agent programming*. Springer, 2005, pp. 3–37.

[10] G. Chance, A. Ghobrial, S. Lemaignan, T. Pipe, and K. Eder. "Investigating Determinism of Gaming Engines for Autonomous Vehicle Verification". In: (Apr. 2020). URL: http://arxiv.org/abs/xxxx.xxxxx.

[11] K. Eder, P. Flach, and H. W. Hsueh. "Towards automating simulation-based design verification using ILP". In: *Lecture Notes in Computer Science* 4455 LNAI (2007), pp. 154–168.

[12] E. P. Enoiu and M. Frasheri. "Test Agents: The Next Generation of Test Cases". In: *2nd IEEE Workshop on NEXt level of Test Automation* (2019).

[13] EURO-NCAP. *EUROPEAN NEW CAR ASSESSMENT PROGRAMME (Euro NCAP) V2.0.4*. Tech. rep. Accessed: 2019-12-13.

[14] M. Fewster and D. Graham. *Software test automation*. Addison-Wesley Reading, 1999.

[15] M. Georgeff. "Communication and interaction in multi-agent planning". In: *Readings in distributed artificial intelligence*. Elsevier, 1988, pp. 200–204.

[16] M. P. Georgeff and A. L. Lansky. "Reactive reasoning and planning." In: *AAAI*. Vol. 87. 1987, pp. 677–682.

[17] S. Hallerbach, Y. Xia, U. Eberle, and F. Koester. "Simulation-Based Identification of Critical Scenarios for Cooperative and Automated Vehicles". In: *SAE International Journal of Connected and Automated Vehicles* 1.2 (2018).

[18] *IEEE Standard Glossary of Software Engineering Terminology*. 1990, pp. 1–84.

[19] C. Ioannides and K. I. Eder. "Coverage-Directed Test Generation Automated by Machine Learning – A Review". In: *ACM Trans. Des. Autom. Electron. Syst.* 17.1 (Jan. 2012), 7:1–7:21.

[20] N. Kalra and S. M. Paddock. "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" In: *Transp. Research Part A Policy and Practice* 94 (2016), pp. 182–193.

[21] P. Koopman and F. Fratrik. "How many operational design domains, objects, and events?" In: *CEUR Workshop Proceedings* 2301 (2019), pp. 1–4.

[22] P. Koopman and M. Wagner. "Challenges in Autonomous Vehicle Testing and Validation". In: *SAE International Journal of Transportation Safety* 4.1 (2016), pp. 15–24.

[23] P. Koopman and M. Wagner. "Toward a Framework for Highly Automated Vehicle Safety Validation". In: *SAE Technical Paper Series* 1 (2018), pp. 1–13.

[24] K. Korosec. *Waymo's self driving cars hit 10 million miles*. https://techcrunch.com/2018/10/10/waymos-self-driving-cars-hit-10-million-miles. Accessed: 2019-11-26.

[25] F. S. Melo. "Convergence of Q-learning: A simple proof". In: *Institute Of Systems and Robotics, Tech. Rep* (2001), pp. 1–4.

[26] F. S. Melo, S. P. Meyn, and M. I. Ribeiro. "An analysis of reinforcement learning with function approximation". In: *Proceedings of the 25th International Conference on Machine Learning*. 2008, pp. 664–671.

[27] T. Menzel, G. Bagschik, and M. Maurer. "Scenarios for Development, Test and Validation of Automated Vehicles". In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. June 2018, pp. 1821–1827.

[28] G. E. Mullins, P. G. Stankiewicz, R. C. Hawthorne, and S. K. Gupta. "Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles". In: *Journal of Systems and Software* 137 (Mar. 2018), pp. 197–215.

[29] *Road Safety Annual Report*. https://www.itf-oecd.org/sites/default/files/docs/irtad-road-safety-annual-report-2018_2.pdf. Accessed: 2019-11-26.

[30] E. Rocklage, H. Kraft, A. Karatas, and J. Seewig. "Automated scenario generation for regression testing of autonomous vehicles". In: *IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. Oct. 2017, pp. 476–483.

[31] Z. Saigol and A. Peters. "Verifying automated driving systems in simulation framework and challenges". In: *25th ITS World Congress* September (2018), pp. 17–21.

[32] *The Highway Code*. https://www.gov.uk/guidance/the-highway-code. Accessed: 2019-11-25. Department of Transport, UK, 2015.

[33] C. E. Tuncali and G. Fainekos. "Rapidly-exploring Random Trees for Testing Automated Vehicles". In: *22nd IEEE Intelligent Transportation Systems Conference (ITSC 2019)* (2019), pp. 661–666.

[34] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski. “Simulation-based Adversarial Test Generation for Autonomous Vehicles with Machine Learning Components”. In: *IEEE Intelligent Vehicles Symposium*. Vol. 2018-June. IEEE, June 2018, pp. 1555–1562.

[35] *UK Road Traffic Act 1988*. http://www.legislation.gov.uk/ukpga/1988/52/contents. Accessed: 2019-10-03.

[36] S. Ulbrich, T. Menzel, A. Reschka, F. Schuldt, and M. Maurer. “Defining and Substantiating the Terms Scene, Situation, and Scenario for Automated Driving”. In: *IEEE Conference on Intelligent Transportation Systems* 2015-Oct (2015), pp. 982–988.

[37] M. Utting, A. Pretschner, and B. Legeard. “A taxonomy of model-based testing approaches”. In: *Software Testing, Verification and Reliability* 22.5 (2012), pp. 297–312.

[38] *Vienna Convention on Road Traffic*. https://treaties.un.org. Accessed: 2019-10-03.

[39] C. J. Watkins and P. Dayan. “Q-learning”. In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[40] A. Zoumpoulaki, N. Avradinis, and S. Vosinakis. “A Multi-Agent Simulation Framework for Emergency Evacuations Incorporating Personality and Emotions”. In: *Hellenic Conference on Artificial Intelligence. Springer, Berlin, Heidelberg, 2010.* 2010, pp. 423–428.