A Multi-Agent Systems Approach to Test Generation for Simulation-based Autonomous Vehicle Verification

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

***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 preconditions of assertions. On a small example we show that, by encoding a variety of different behaviours respondent to the agent's perceptions of the test environment, the agent-based approach generates twice as many effective tests than pseudo-random test generation. It is both scalable and efficient. Moreover, agents can be encoded to behave more naturally without compromising the effectiveness of test generation. Our results suggest that generating tests using testing agents significantly improves upon random and simultaneously provides more realistic driving scenarios.**

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

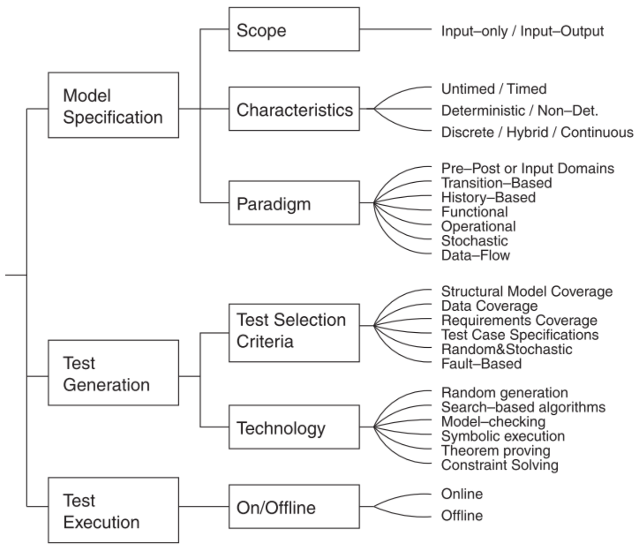

Fig. 1. Taxonomy of model-based test generation, from [**utting2012taxonomy**]

I. Introduction

Verification is the process used to gain confidence in the correctness of a system with respect to its requirements [**bergeron2012writing**]. 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 [**utting2012taxonomy**].

Testing autonomous driving functions and safety critical scenarios in simulation can benefit from a fully controlled environment where road layouts, weather conditions and other driving scenario parameters can be directed to achieve specific test targets. These tests may look to convince auditors of the functional safety of the vehicle or whether it complies with commonly agreed upon road conduct, such as the Vienna convention [**ViennaConv**], interpreted locally by each country as a set of rules [**codes2011highway**] and road traffic laws and penalties [**RoadTraffic1988**].

Verification of autonomous vehicle face the same challenges as other complex systems, e.g. semiconductor hardware, where it has been recognised that verification can take up to 70% of the design effort [**arden2002international**] and exhaustive testing is intractabe due to vast parameter space. Estimates vary but demonstrating autonomous vehicle safety with a 95% confidence 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 [**kalra2016driving**]. This figure is based on the number of people killed on roads in the US and is therefore somewhat arbitrary as the criteria will therefore be proportional to overall driving safety in each country, e.g. UK road fatalities per billion vehicle-km is half that of the US in 2018 [**ITFroadSafety2018**].

So is there a way to achieve confidence in the design without millions of miles of driving or billions of miles of simulated driving [**korosec2019waymo**]? Furthermore, is on-road testing really suitable if the goal is to ensure the AV handles rare events appropriately when such events will be witnessed inefficiently by chance [**Koopman2018**]?

Considering the AV as a device under test (DUT) the verification engineer is presented with a familiar set of tasks entailing the discovery of bugs, corner cases, unanticipated events and to reach complete coverage. A suitable analogy of the autonomous vehicle in this context is that of a responder which interacts with an abstracted level of the testing envi-

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

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

ronment and only interacts passively (responds) to a recieved stimulus and whilst also fulfilling it's own objectives is also expected to adhere to legal protocol, which are the expected driving behaviours in response to that stimulus. Investigating the opportunity of *agency* to address the challenges of verifying the responder is the principle hypothesis of this research. The task of this software *agent* then becomes targetted with specific goals to find the corner cases, rare events and ensure coverage against traffic regulations which can be encoded in the behaviour of the agent with respect to the coverage point. Our key research question is: can agent based test generation be used as a form of simulation based verification for autonomous vehicles that will satisfy the requiements without millions of miles of testing? We aim to show that an agent based test generation method can be considered a potential route to solving the AV verification problem and that test cases generated are better than random.

How will this method be considered successful? For this method to be considered a success it must produce stimuli to the responder (simulated AV) that satisfies the condition of a 'good' test case. A good test case should be *effective* (at detecting defects), *efficient* (minimising the number if tests), *economic* (cheap to perform and analyse) and *evolvable* (adaptable to software changes) as defined in [**fewster1999software**]. This is explored further in the case study in Section III.

The agent based test generation technique will sit alongside and complement the techniques in the existing verification taxonomy [**utting2012taxonomy**]. The new method will form a new leaf node at the Test Generation-Technology branch of the existing taxonomy of model-based test generation, Fig 1. This work proposes using an agent based model for test generation. This fits in within the existing taxonomy of model-based testing [**utting2012taxonomy**] under the test generation - technology branch.

more/stronger motivation - why agents? what is the challenge/problem?

mention how there are formal and informal rules of the road, both of which are valid tests Regulators would use the former to assess the safety case for the vehicle, whereas public opinion may be persuaded more by compliance of the latter. Trustworthiness will certainly be composed of both aspects of differing proportions depending on the audience.

The proposed test bench, 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, execution conditions for an item to be tested [**StandardsBoard1990**] to the test bench. The Vehicle Behaviour Interface (VBI) connects the AV controller 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.

As well defined in [**Ulbrich2015**]; Scene referes to all static objects including the road network, street furniture and a snapshot of any dynamic elements. Any dynamic element

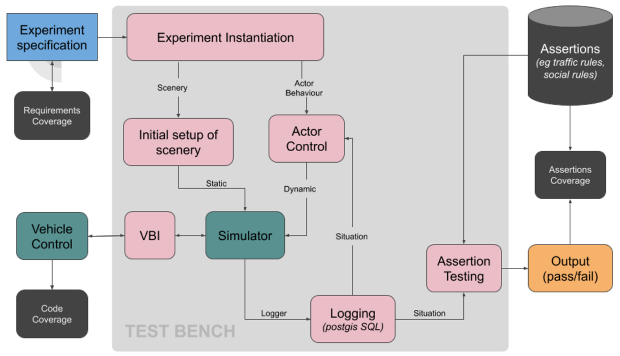

Fig. 2. Test bench design.

whose actions or behaviour may change with respect to time are considered actors which may include other road vehicles, pedestrians and traffic signals. A situation is defined as the subjective conditions and determinants for behaviour at a particular point in time. The scenario is defined as a temporal development between several scenes which may be specified by specific goals and values.

The test case specification may be generated in a number of ways familiar to the verification engineer. Tests may be manually generated which will be both valid and accurate in producing the desired test conditions but at the highest cost. Random methods are usually employed at an early stage of testing to get coverage quickly but potentially suffer from generating invalid tests or tests that are not interesting, where interesting in this case refers to exercising the decision making processes of the AV. Model-based test generation sits between these methods in terms of validity of tests and the cost to produce them. Model-based testing requires a model that encodes the behaviour of the test environment then used to generate tests for execution. The proposed method focuses on the development between scenes dictated by the goals or values of the dynamic actors in the scene. A suitable framework for this appraoch to control is the Multi-Agent Systems paradigm.

A. Multi-Agent System

Georgeff and Lansky were key in the development of the belief-desire intention (BDI) agent programming approach [**georgeff1987reactive**] and also in the early work of multi-agent systems [**georgeff1988communication**]. Automatically generating stimulus that will increase functional coverage is a key challenge in simulation based verification and is a key vitue of constrained pseudo-random techniques. But such techniques are also far from completely automated, as the verification engineer must supply the suitbale constriants which become ever more complex in real driving scenarios. Feedback based coverage-driven generation (CDG) employs machine learning techniques to automatically generate tests based on rewards from a neural network or Markov Decision Process (MDP). Inductive Logic Programming based Coverage Driven Generation (ILP-CDG) has been shown to improve coverage closure by training on data generated during psuedorandom test generation and learning rules to constrain subsequent simulations [**Eder2007**]. Combining the BDI framework with an automated test generation approach leads to the idea of a software agent capable of generating tests. Intelligent agents have been used in the human robot interaction (HRI) domain with a coverage feedback driven generation approach using reinforcement learning to explore collaborative manufacturing [**Araiza-Illan2016**]. The idea of a test agent has also been proposed by Enoiu et al., [**Enoiu2019**] for regression testing although more for test selection from a library rather than generation, where agents decide what tests to execute and with what prioritisation and includes inter-agent messaging. Hawkins et al. [**hawkins2019situation**] use the Multi-Agent Simulator Of Neighbourhoods (MASON) platform to filter situation coverage from a small model for an autonomous robot (AR) to efficiently ensure parameter combinations are not repeated.

II. Related Work

A good overview of the challenges currently faced in software testing of autonomous vehicles are well document by Koopman highlighting the importance of fault injection into the testing domain [**Koopman2016**] and how to decide what aspects of the system need to be tested in the areas of operational design domains, event detection and vehicle maneuvers [**Koopman2019**]. Describing a driving scene in natural language has been shown to be an effective framework for scenario generation which can also be automated based formalised rules and knowledge [**Bagschik2018**] or even evolving from ISO safety standards [**Menzel2018**]. Hallenbach et al. take the approach to test generation of developing composite metrics for traffic quality and using them to determine if a senarios is 'critical' or not and therefore worth exploring further [**Hallerbach2018**]. Generating targeted test cases can be a more efficient way to get complete coverage. By finding tests that exercise the complete decision making domain we can do away with the 'unnecessary miles' of testing precisely explored by Mullins et al., [**Mullins2018**] where they describe a test generation method based on the transitions between decision making performance modes of the AV. Saigol et al. [**Saigol2018**] discuss the need for *smart actor control* which may lead to interesting scenarios when interacting with the AV in their proposed test bench framework for automated testing. Rocklage et al. [**Rocklage2017**] approach the problem of test generation from the veiwpoint of the other traffic participants, using a trajectory planner to ensure coverage over a fixed list of scene parameters.

A. Natural Behaviour vs. Edge Case

Natural = waiting for interesting event, Edge case = find only interesting events. 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. In this case such efforts seek to reduce the *reality gap* [**Jakobi1995**] providing the most likely scenarios and actor behaviour. One approach is to monitor real traffic scenarios from traffic monitoring cameras, tracking individual vehicles, cyclists and pedestrians and from these build behavioural models for each road scene for which you have traffic data [**latentlogic**].

This research proposes that agent based test generation can be used as an efficient and scaleable method for AV verification. Within existing multi-agent frameworks, e.g. JASON [**bordini2005jason**] agents are given specific goals and actions which are updated based on perceptions of the environment. The proposed implementation would work similarly, where agents (cars, pedestrians, cyclists, traffic lights etc.) are given goals that relate to the coverage task required. As such, simulated driving scenarios are not prescriptive at the outset but evolve with time based on the actions of the test vehicle.

In this paper we explore the use of a multi-agent system approach to test generation. In theory the MAS technique should achieve better accuracy than a random method as agents can be encoded with certain behaviours that drive the scenario towards a precondition for the assertion that needs to be tested. This should inccur a relatively small cost on the development engineer due to the abstraction of goals in agent based programming but at the cost of additional lines of code and resources used at runtime. A small example is used to illustrate these metrics in a single scene that uses a MAS framework, e.g. Jason [**bordini2005jason**], to generate interesting tests. A full description of the assessment is given in Section **??**. This is a conceptual proof and not a complete verification process, choosing only a single assertion may is limited but the goal here is to prove that with a small economic outlay of engineering time we can generate agents that outperform random.

III. Case Study

This section describes a case study to explore the proposed test generation method. The aim is to generate tests that exercise the precondition for a single assertion: collision avoidance. The precondition in this case states that for the associated decision making process in the AV to be exercised an entity in the path of the vehicle should be avoided either by braking or manouvering. A single assertion is chosen to isolate assessment of the test generation concept and not a demonstrate a complete verification process. The assertion chosen in this example is collision avoidance with a pedestrian and a successful test would be when the pedestrian enters the emergency braking zone of the DUT, i.e. within a 1s time-to-collision (TTC) window of the leading edge of the vehicle. The outcome of the test and the behaviour of the AV are not the focus here, but that tests are generated using the agent framework that satisfy the precondition for the assertion.

A. Test Agents

To assess if the agent based model of test generation is effective it is compared to random methods and repeated sufficiently for statistical power. A simple scene is chosen containing a straight road with a pavement either side represented in a grid world with a finite (1.0s) time interval. For this example pedestrian and test agent are synonymous. At the start of a test pedestrians are randomly located onto the pavement area and the AV is represented as a vehicle that drives at constant speed in a straight line and will not brake or turn. The test agents have differing levels of behaviour from simple random to more complex and directed, which is the focus of this section.

The different pedestrian actions can be split into two main classes: random and directed, see Table I. For the random class, the *random action* is where the test agent can make any random movement at each tick where the available actions are: do nothing (stand still), move up, down, left or right. For the *random behaviour* the pedestrian is instantiated walking along the pavement before the simulation starts and has only one action; to randomly choose when to cross the road. The random behaviour is included so that a comparison between crossing the road at a targeted or random time can be made.

The second class of actions are based on perceptions of the pedestrian and these agents are initialised walking along the pavement as in random behaviour. The *proximity* behaviour instructs the agent to cross the road when the AV is within a certain radius. Using the *intersect* behaviour, the agent calculates an intersection time and chooses to cross the road if the pedestrian can reach the intersection point before the AV has passed. The *election* behaviour ranks the probability of each agent to enact the intersect behaviour and elects the agent with the highest likelihood of success.

The *proximity* behaviour describes an agent, for this assertion, with suicidal tendencies that will step into the road whenever a car is nearby but is a highly improbably model of pedestrian actions. The *proximity* behaviour also suffers from the 'zombie effect' where all nearby pedestrians are drawn to the passing vehicle which may lead to the coverage point required but through an improbable scenario. The *intersect* behaviour is a more refined version of the proximity behaviour but may still result in multiple agents attempting to intersect the AV simultaneously whereas the *election* behaviour should limit this to a single agent.

The number of agents is also a factor to consider for this demonstration, too few and the likelihood of an agent triggering the assertion will be low and entirely dependent on the initial starting position in the test environment due to differences in speed. The maximum number of agents could be considered all the available grid locations (1.5m spacing) on the pavement without overlap. As the number of agents, nA, tends to this maximum the probability of hitting the coverage point using a random class increases significantly as more agents appear in the road with an increase in computation expense. As nA increases using more intelligent behaviour methods, suitable agents to complete the assertion should be found more readily resulting in shorter and hence more efficient tests. The range of nA explored was 1-10.

As with any model refinements could be made to improve coverage success, such as comparing direction of travel between the AV and the agent or modifying pedestrian speed, but this example is kept simple to illustrate the concept.

B. Test Environment

The testing environment, see Fig. 3 is a straight two-lane road 99m long and each lane is 6m wide. Pavements are on

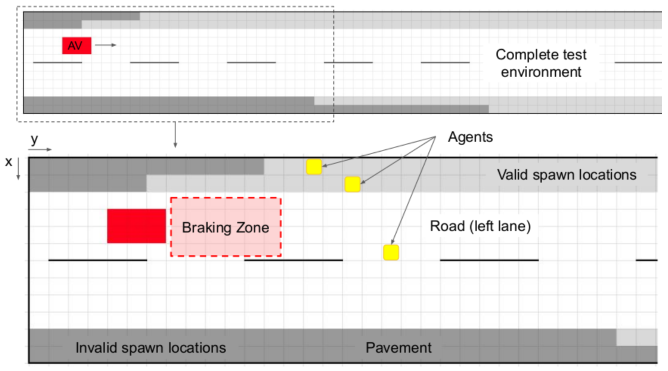

Fig. 3. Test environment at full scale (left) and in detail (right) including valid and invalid spawn locations for the pedestrians and the position and direction of the AV.

both sides of the road which are three meters in width giving a total size is 18m x 99m. The assumed AV velocity is 20 mph which is equivalent to 9.1 m/s that is rounded down to 9 m/s. The pedestrian velocity is 3 mph equivalent 1.4 m/s rounded up to 1.5 m/s. The map is discretised with 1.5 meter resolution for simple division into the AV and pedestrians velocities. Thus, in the discretised world the AV velocity is 6 unit/sec and the pedestrians velocity is 1 unit/sec. The total number of map cells is 12 x 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 assertion is triggered or the AV reaches the end of the road then the test is restarted. The AV occupies an area 4.5x3m equivalent to 4x2 cells. There are no other vehicles and the right hand side of the road is unoccupied.

The environment is initialised with the agents spawned randomly on any valid pavement area (see note below). When the environment is reset new random locations are set for the agents. Control over random locations are set using a fixed seed based on the experiment number (1-1000) and so tests are repeatable between agent types, i.e. An experiment with 5 agents using a random method with have the same starting locations as using any other method which ensures valid comparison between agent types.

Valid test if agent found in 'braking zone' of the ave which extends 6 units forwards of the leading edge of the vehicle. Any agent found in this area is defined as a valid test and the environment is reset.

Some areas of the environment are impossible for the agents to intersect the AV and are therefore invalid spawn locations. On the left hand-side of the left pavement a pedestrian needs to move three cells to the right to reach the left part of the centre of the left lane. This is equivalent to three seconds. In this time the AV crosses 18 units. Therefore, the first 18 cells of the left-hand side of the left pavement are dead-zone where pedestrians cannot intersect the car if spawn inside this zone. Similarly, the first 12 cells of the right-hand side of the left pavement, the first 36 cells of the left-hand side of the right pavement and the first 54 cells of the right-hand side of the right pavement are all invalid spawn locations.

C. Scoring

Each agent behavioural method will be repeated 1'000 times and the number of valid tests counted. To reiterate, a valid test is when a pedestrian intersects the AV and for simplicity we assume this is when they have the same coordinates. In an attempt to impose more realistic behaviour on the agents, a scoring system is used that penalises certain actions. A living cost is imposed on the agents to promote shorter tests and a penalty is given for agents that are in the road but not intersecting the AV.

The scoring system is:

- A reward of 20 is given if the agent intersects the AV.
- A penalty of -1 for each time step.
- A penalty of -7 for each time step spent in the road.

D. Simulation and Logging

Each of the agent behaviours were written into the Jason platform along with the test environment as described above. For each test a basic log of the agent actions, score and time to completion are recorded and then averaged for each type. A list of random start locations was created for the pedestrians and this was done for each value of nA explored, i.e. 1-10. The list was then used to spawn the pedestrians for each behavioural type to ensure the initial conditions were identical.

Ultimately the scoring should reflect the benefits of the methods against the needs of test generation: accuracy, cost and robustness. We want tests that are accurate (valid tests, collects coverage) and sufficiently low cost (CPU hours, engineering time). If tests are robust they are adaptable to real-time events, e.g. AV slows down around bend requiring agents to recalculate intersection.

IV. Results

The results section is split into 4 parts; Accuracy is the ratio of sucessfull tests the agents generated as a ratio of all tests, Score is a measure of how natural the agents behaved, Combined Score combines accuracy with score, and Time is how long the agents took to generate the test in both simulation ticks and wall clock time. Both Score and Time have distributions associated with them and as such confidence intervals are provided.

A. Test Accuracy

Test generation accuracy, defined as the number of tests that have activated the precondition for the asserion as a ratio of all tests, are shown for each agent type in Fig. 4. The *random action* (RA) and *random behaviour* (RB) have lower accuracy compared to directed agents when $nA < 10$. For high nA the accuracies converge mostly due to a saturation of agents (explained above). Fig. 4 also shows that for $nA = 1$ a directed agent outperforms a random agent by more than 2:1. The *election* agent generates a slightly higher accuracy than the *proximity* agent but only by 2.2%. However, for $nA = 2$ and above the *proximity* agent has a 10% increase over the election agent which is because this agent has multiple attempts to trigger the precondition whereas the election agent only has one.

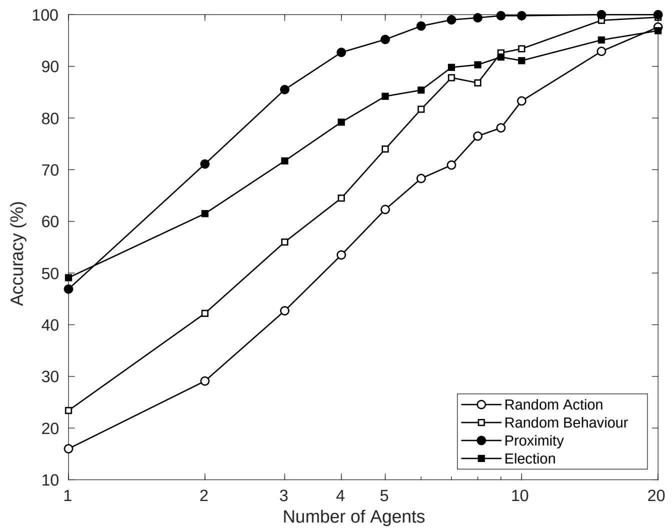

Fig. 4. Accuracy.

B. Test Score

For tests that activate the precondition the average agent score is compiled including 95% confidence interval range, see Fig. 5. The maximum theoretical score of any agent is 94 which includes 100 points for a successful test sub tracting a living cost of 1 and road penalty of 5. For a single agent scores between agent types are similar as accuracy is being controlled for by including only the successful tests. As the number of agents increases the random class diverge from the directed class and the variance in the random agent score increases.

As nA increases for the random agents, the number of agents found on the road also increases and hence average score drops rapidly, whereas the directed class are only found crossing the road when they deem fit and are on the pavement at all other times keeping the score much higher. The high score for the $nA = 1$ random class is surprising especially as the error bars are small indicating that, although the accuracy is low, tests generated are relatively efficient, i.e. the pedestrian does not spend a lot of time in the road. No significant dfference between the directed class is seen in the score even with a high number of agents indicating both represent a similar level of natural behaviour, at least within the limited scope of this example.

C. Combined Score

As the score (above section xx.xx) controls for accuracy it can be misleading to interpret these results so a combined score is provided which is given by 1x10-3 * score * accuracy. This combined measure promotes scores that are attached to high accuracies, describing agents that can generate useful tests with natural pedestrian behaviour. Time could have been included as a numerator here but is similar in scope to the living penalty so not included.The normalised results, Fig 6, show that the directed agents are over twice as effective within this new combined definition than random agents for $nA = 1$ although this advanted drops rapidly with increasing agent numbers to reach a steady gap of around 12%. The *random*

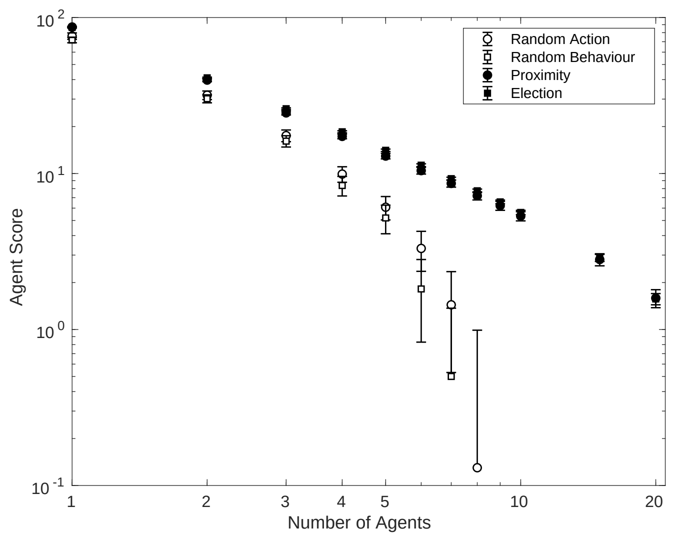

Fig. 5. The average agent score for successful tests with 95% confidence intervals for each agent type. Random class are hollow markers and directed class are filled. The maximum theoretical score is 94 points but this is only applicable to a single agent hence the declining average with increasing agent numbers.

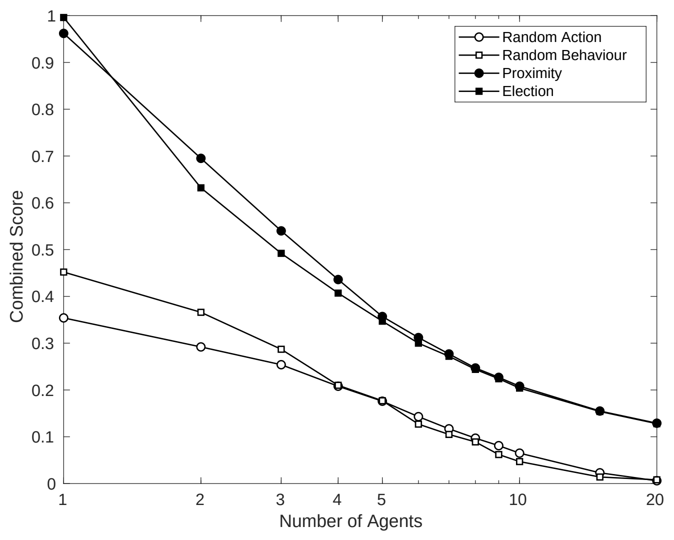

Fig. 6. The combined accuracy and score for each agent type.

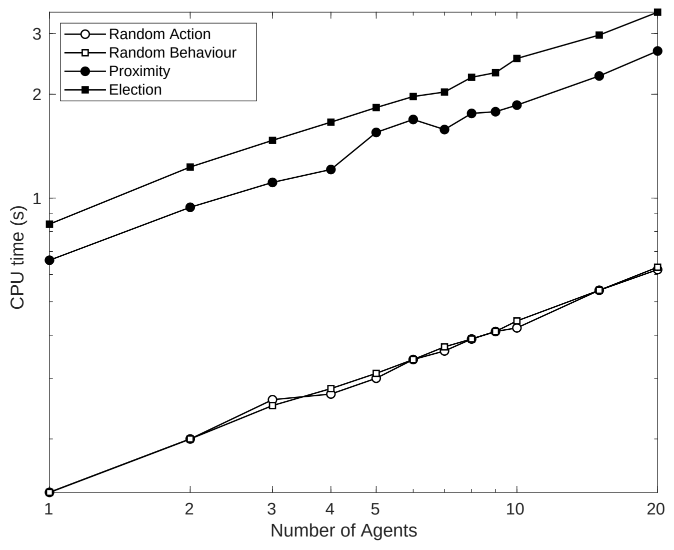

Fig. 7. The CPU time taken to execute the agent actions averaged over 1000 tests. The more complex agents have additional CPU overhead compared to random but this difference is relatively unchaged with agent numbers.

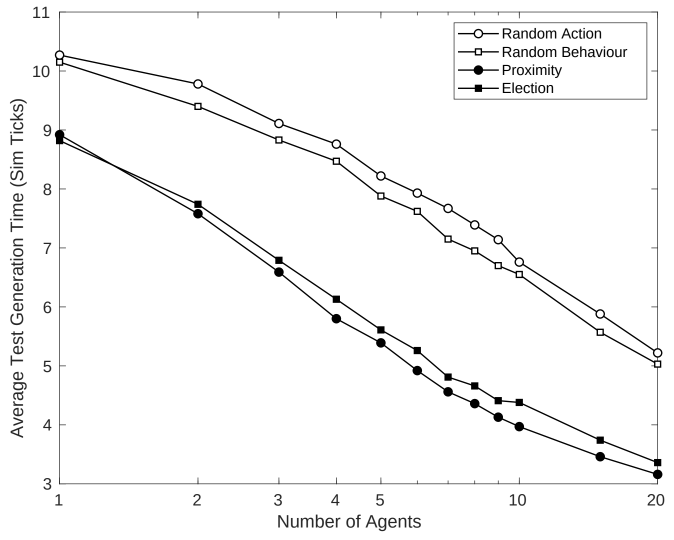

Fig. 8. The average time taken for an agent to find a successful test.

behaviour agent outperforms the *random action* for $nA < 4$ beyound which there is little difference. So if random is your choice, you might as well just use completely random unless you want to use low agent numbers. The *election* agent has the highest combined score for $nA = 1$ but higher agent numbers are favor the *proximity* agent up to $nA = 4$ beyond which there is no notable difference between the directed agent types.

D. Agent CPU Time

Metric of efficiency of the method, resource useage.

For each test generated the met the precondition, the CPU time taken to execute the agent action was averaged over the k-tests and compared across different agent types and numbers, Fig. 7. This is essentially how much additional CPU time is required to execute the actions of a random agent over a more commplex one.

E. Time to generate test

For each successful test generated the reported time was averaged over k-tests and compared across different agent numbers, Fig. 8.

The directed agents consistently improve over the time taken by the random class for $nA = 1$ by around 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 random agents. Overall the *proximity* agents find tests in the shortest number of simulation steps.

TABLE I
TEST AGENT SUMMARY TABLE AND SAMPLE OF RESULTS FOR $nA = 3$.

Method	Description	Accuracy	Combined Score	Code Lines	CPU time (s)
Random Action	Randomly perform action (up, down, left, right, stop).	42.7	0.254	23	0.09
Random Behaviour	Walk along pavement, randomly cross the road	56.0	0.287	82	0.08
Proximity	Walk along pavement, Cross road when AV in range	85.5	0.540	86	0.37
Election	As in Proximity but elect a single agent to cross	71.7	1.470	235	0.49

F. Results Analysis

discuss how the test show compliance against the 'good test case' description in the intro: Accuracy is not precisely detecting defects in this case but the AV is put inot a situation where this is tested, Agent CPU is a resource metric = ecomonic, TTGT also economic but more a 'return on investment' with efficiency. Agent score can be considered as the evolvable attribute as agents with a high score should do well in other environments (road layouts here) and still maintain realistic behaviour. The only attribute not covered is exemplary as in this case study only a single assertion is considered.

Agent density vs. accuracy - can be used to det how many agents required for map size

The results show that by nearly all the metrics and parameter combinations discussed above, (accuracy, natural behaviour score, test generation time) the directed agents outperform the random agents. This shows that even a small amount of intelligence can be a distinct advantage over random techniques. But what was the cost of developing these agent behaviours? And is there a limited payoff to gain significantly superior intelligence for really complex agent behaviour? Comparing the lines of code for the agent action definition can help to weigh the investment in agent complexity. Lines of code for each are below and the combined score for $nA = 1$ is in brackets:

Remove this list and refer to the table

- *random action* = 23 (0.35),
- *random behaviour* = 82 (0.45),
- *proximity* = 86 (0.96),
- *election* = 235 (1.0)

indicating that for a small investment the *random action* agent can be improved significantly with 4x the initia code to improve it's combined score to near the theoretical maximal. To improve further with the *election* agent took 10x the initial code with minmal score improvement suggesting this was not worth the investment and also considering for $nA > 1$ *proximity* proved more effective.

This also raises the question of how comlex the agents sould be? The analysis above would suggest that the level of agent complexity should be considered carefully as a simpler level of intelligence could be more beneficial.

V. Conclusion

The MAS programming paradigm offers rational agency and strategic planning to software agents that have been exploited in this research for test generation. On a small example we show that, by encoding a variety of different behaviours respondent to the agent's perceptions of the test environment, the agent-based 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 significantly improves upon random and simultaneously provides more realistic driving scenarios.

A. Future Work

In this small case study there is only a single assertion considered. But how agents behave when multiple assertions or *desires* exist will be key to the success of this method. This could be achieved with a number of methods such as MDP and ANN. As discussed in [**Eder2007**] agents that have their goal selection modified based on coverage feedback would be an important step in this direction. This also fits in with the feedback reward of many such architectures, e.g. rewards for MDP [**littman1994markov**] and backpropagation for ANN [**foerster2016learning**].

Composing agents with a feature based representation of their environment is also a direction to ensure agents can scale to large physical maps and also adaptable to new features as needed. This approach also fits in with most ML techniques. Including personality to agents is also another avenue that could provide insightful as a tuning parameter [**Zoumpoulaki2010**]. Conventionally agressive (or egoist) behaviour may seem the most interesting as edge cases and therefore require oversampmling. But there could also be merit in exploring more risk-advers behaviours or simply in understanding what normative driving conditions are as a baseline for CAV cyber security. There is also the possibility to have have combination of random and directed agents to ensure the initial coverage is satisfied, then add more intelligent agents for hole coverage.

Code used in this analysis is available at https://github.com/TSL-UOB/CAV-MAS.

Acknowledgment

The ROBOPILOT and CAPRI projects are part-funded by the Centre for Connected and Autonomous Vehicles (CCAV), delivered in partnership with Innovate UK. It is part of the governments 100 million Intelligent Mobility Fund, supporting the Future of Mobility Grand Challenge. (Innovate UK, Grant no.s 103703 and 103288 respectively).

References