Automated Scenario Generation for Regression Testing of Autonomous Vehicles

Elias Rocklage[1], Heiko Kraft[2], Abdullah Karatas[3], Jörg Seewig[4]

***Abstract*— Autonomous vehicles are technologically feasible and are becoming a reality. However, before they can be launched, thorough testing is necessary. In this paper, we present a novel approach to automatically generate test scenarios for regression testing of autonomous vehicle systems as a black box in a virtual simulation environment. To achieve this we focus on the problem of generating the motion of other traffic participants without loss of generality. We combine the combinatorial interaction testing approach with a simple trajectory planner as a feasibility checker to generate efficient test sets with variable coverage. The underlying constraint satisfaction problem is solved with a simple backtracking algorithm.**

I. INTRODUCTION

Autonomous vehicles are becoming a reality. Technological advances resulting from extensive research in areas such as behavior-, motion planning, situation analysis, sensors, perception, sensor fusion and machine learning make it possible. All those fields are combined with each other to yield a complex software system that can perceive, process and interact with its environment. However, compared to the aforementioned fields, astonishingly little work has been done on the testing and validation side and even less research can be found that specifically investigates the topic of regression testing of such systems. The problem of validating and testing complex, intelligent systems that have to handle vast amounts of input data to interact with other systems is very tough. This is mainly due to an infinite and time dependent *input domain* (cf. Def. 8) that can also be influenced by the actions taken by the autonomous vehicle itself. Therefor, as key contributors pointed out, if testing and validation methods cannot keep up with the functional development, they might become the "bottleneck" for the release of autonomous vehicles [1], [2], [3]. To make autonomous vehicles safe and reliable, which is a crucial requirement for a wide adoption, manufacturers need to test these vehicles very thoroughly. Testing on real roads is inevitable and necessary but not entirely sufficient. Challenging factors are the high number of miles that would have to be driven due to the low likelihood of certain situations occurring and the cost/time that is associated with the tremendous driving effort [3]. Moreover reproducibility of certain difficult situations constitutes a problem that makes regression testing nearly impossible. These disadvantages of real road testing can be overcome by testing in a virtual simulation environment. Re the latter, this question has to be answered: What is to be tested and, more specifically, which inputs have to be passed to the system under test.

[1]Elias Rocklage is with Mercedes-Benz R & D North America, 94085 Sunnyvale, CA, United States. eliasrocklage@gmail.com

[2]Heiko Kraft is with Mercedes-Benz R & D North America, 94085 Sunnyvale, CA, United States. heiko.kraft@daimler.com

[3]Abdullah Karatas is with University of Kaiserslautern, 67663 Kaiserslautern, Germany. karatas@mv.uni-kl.de

[4]Jörg Seewig is with University of Kaiserslautern, 67663 Kaiserslautern, Germany. seewig@mv.uni-kl.de

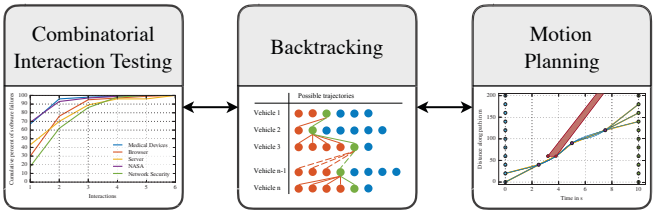

Fig. 1. Main components of the proposed algorithm: To achieve scalable scenario domain coverage with a minimum number of tests, a CIT algorithm is used. Planning the motion of dynamic objects in combination with the CIT algorithm, yields a constraint satisfaction problem that is solved by a backtracking algorithm.

Our first contribution is the definition of different types of traffic scenarios, together with further terminology for regression testing in a virtual environment (Sec. III-A). Those definitions provide a basis for further discussions about the type of scenarios that should be used for regression testing of autonomous vehicles, as will be discussed in Sec. III-B and IV. After the type of necessary scenario is determined, we use the four-layer concept proposed in [4] as a basis for a new method that can be used to create *static* (cf. Def. 9) or *hybrid* (cf. Def. 11) scenarios while guaranteeing a certain coverage of parameter combinations (Sec. IV and V). In this second contribution, we focus on solving a multi-vehicle motion-planning problem without the loss of generality with respect to static objects and other time-invariant parameters. The novelty of the approach is justified by the fact that a combinatorial interaction-testing algorithm (Sec. V-A) is used together with a backtracking algorithm (Sec. V-B) and a motion planner (Sec. V-C) that, in this composition, serves as a feasibility checker.

II. RELATED WORK

In [4] the authors explain how current advanced driver assistance systems (ADAS) are tested and that their area of operation is mostly outside of urban areas. They explain the need for more test scenarios and the growth of complexity as situations take place in urban environments. First, the authors propose a four-layer concept to identify variable parameters that influence the system under test (SUT), whereupon those

978-1-5386-1526-3/17/$31.00 ©2017 IEEE

parameters are used to generate scenarios using combinatorial methods and blackbox testing techniques. Combinatorial testing methods look promising since they enable the tester to create compact test sets with variable parameter combination coverage. The idea behind combinatorial interaction testing (CIT) is the empirical finding that most software failures are based on only a few interacting parameters [11]. However, the publication [4] only describes the entire concept on an abstract level and does not go into detail on how the systematic test generation works exactly. They do not describe how the different layers of the parameter identification phase can be tested with each other and how this affects the number of test cases. To be more precise, they do not mention if the combinatorial methods are used to combine the parameters of each layer separately or as a whole. If the former is the case, the proposed concept would scale exponentially. Assuming that each layer consists of 10 mutations that where derived using combinatorial methods, the total number of test cases would yield $10^4 = 10000$, which is quite a large number, taking into account that 10 variations per layer is a very conservative estimation.

Given the vague explanation mentioned above, the publication does not go into enough detail to comprehend how exactly *dynamic objects* (cf. Def. 2) are defined and handled during a test run.

The information that the motion of the vehicles can be defined through parameters relative to the ego vehicle, which in this case represents the SUT, indicates that the other traffic participants also have some form of artificial intelligence that controls them. This artificial intelligence must be configurable with different parameters that in turn can be used for the systematic test generation.

Since the illustrations presented in [4] look like Virtual Test Drive (VTD) by Vires [13], it further can be assumed that the traffic is generated by using the "Pulk Traffic" feature of VTD, which is widely configurable [12]. The problem with this method of traffic generation is that it heavily relies on the implementation of the autonomous driver. Even though different parameters are adjusted, it is hard to ensure that, for a given base route and a given situation dependent adaption of that route, all possible traffic constellations are covered. In the case of the autonomous driver in VTD for example, the implementation does always show *cooperative behavior* (cf. Def. 4), which excludes all situations from the test set where the ego vehicle is supposed to be forced into a situation (cf. Appendix VI-A).

III. VIRTUAL REGRESSION TESTING

In literature how regression testing in a virtual environment such as VTD can be conducted has not been investigated yet. Dealing with this topic requires a distinct definition of some terminology as has been discussed in [14] and [15]. However those definitions are not targeted entirely for purely autonomous vehicle testing and do not cover some terminology that is important for regression testing of such systems. Therefore some basic definitions are provided below, and these definitions are then used to create a problem definition for regression testing of autonomous vehicles.

A. Scenario-Specific Terminology

Definition 1. *Ego Vehicle*
The ego vehicle is the autonomous system, that represents the system under test (SUT).

Definition 2. *Dynamic Object*
Dynamic objects are in general objects that change their state gradually and therefore are time dependent (e.g., moving vehicles, but also traffic light phases).

Definition 3. *Static Object*
Static objects are objects that do not change their position over time. A vehicle, for example, that does not move during the entire timespan of a *scenario* (cf. Def. 6).

Definition 4. *Cooperative Behavior*
Dynamic objects (cf. Def. 2) that react to the motion of other vehicles during simulation. Cooperative vehicles, for example, will make space for another vehicle to merge. They anticipate what other traffic participants are going to do and cooperate in friendly manner.

Definition 5. *Noncooperative Behavior*
Opposite to *cooperative behavior* (cf. Def. 4) these dynamic objects only react to other vehicles to avoid a crash. Since they are not pleasant, they will not make space for other vehicles.

Definition 6. *Scenario*
A test case for an autonomous vehicle. It consists of a road section (i.e., an intersection) and static and/or dynamic objects together with the ego vehicle. For regression testing, a *static scenario* (cf. Def. 9) needs to be distinguished from a *dynamic scenario* (cf. Def. 10) and a *hybrid scenario* (cf. Def. 11).

Definition 7. *Scenario Space/Domain*
A *scenario* (cf. Def. 6) can be described by parameters. Those parameters span a high-dimensional manifold that we call *scenario space*. By varying the scenario space, different scenarios can be generated.

Definition 8. *Input Space/Domain of the Ego Vehicle*
The input space of the ego vehicle is a time-dependent, perspective view of the ego vehicle of the *scenario* (cf. Def. 6) and therefore depends on the ego vehicle's motion relative to the scenario.

Definition 9. *Static Scenario*
This is a scenario where *dynamic objects* (cf. Def. 2) follow a predefined trajectory. Dynamic objects do not take other traffic participants like the ego vehicle into account. They stubbornly follow their trajectory even if they crash into another entity. The trajectories are defined before running the simulation. This testing mode can be compared to an open-loop system since the behavior of the ego vehicle does not have an impact on the system.

Definition 10. *Dynamic Scenario*
This is the opposite of the *static scenario* (cf. Def. 9). Dynamic objects take other traffic participants into account. They can show *cooperative* (cf. Def: 4) or *noncooperative* (cf. Def. 5) behavior. Trajectories can be defined explicitly in the the form of path shapes or implicitly as paths where a start and a target position are determined. Their motion is calculated online during simulation with respect to the ego vehicle. Dynamic objects in the context of dynamic scenarios may have a configurable driver model. Following the control terminology from above, dynamic scenarios can be considered a closed-loop system.

Definition 11. *Hybrid Scenario*
This is a mixture of the *static scenario* (cf. Def. 9) and the *dynamic scenario* (cf. Def. 10). The dynamic objects stubbornly follow their predefined trajectories until the point in time when they need to deviate due to the influence of the ego vehicle in order to avoid a crash. From that moment on, the scenario is dynamic.

B. Problem Statement

Regression testing has become a standard technique in software testing and is usually used to compare two software versions [5]. In this manner, confidence can be gained that new features or bug fixes do not interfere with the unchanged parts of the software [5]. In order to use regression testing, test cases are necessary that challenge the SUT in the same way every time the tests are executed. The implications of this statement are as follows.

1) Dynamic Scenarios: If the scenarios are dynamic, two problems arise. First, the behavior of the dynamic objects, even though parameterizable, is subject to the underlying implementation of the driver model. Second, failures might be time dependent which means, that they only occur if a certain sequence of events happens [11]. For regression tests, those sequences need to be the same every time the test is run. In the case of dynamic scenarios, where only the starting state and some vehicle goals are defined, control over sequence coverage is lost. Hence it cannot be ensured that certain sequences of events have been covered.

2) Cooperative Behavior: For regression testing, the cooperative behavior of dynamic objects is problematic since it is dependent on the ego vehicle. In addition, cooperative behavior always simplifies the situation for the ego vehicle by opening up solution spaces that are not subject to the test case that actually aims to confront the SUT with a scenario that is hard to solve.

3) Input Space of the Ego Vehicle: For simple systems, regression testing is simple: For each new software version, the same inputs can be provided to the SUT and the outputs can be evaluated. Unfortunately, the inputs of the ego vehicle depend on the ego vehicle's motion and constitute a perspective view of the scenario. If, after an update of the software, the ego vehicle behaves slightly different, it's motion deviates from the motion of the ego vehicle from the previous test run. Due to this shift, the input of the SUT changes, since the ego vehicle now observes the scenario from different positions. The different inputs might lead to different actions, which means that the SUT is not challenged by the same inputs anymore.

4) Scenario Evaluation: Generating test scenarios is only the first step in testing autonomous vehicles. The test runs also have to be evaluated in order to yield results. Proper evaluation demands that test scenarios last for a certain amount of time. An action taken by the ego vehicle at $t = 1s$ might lead to a failure at $t = 5s$. Due to this fact, the simulated scenarios cannot be arbitrarily short to avoid the event sequence difficulty mentioned above.

5) Number of Scenarios: As already mentioned by other authors, the number of possible scenarios is infinite [4]. This also applies to the scenarios specifically generated for regression testing. However, as [4] points out, different blackbox testing techniques like boundary value analysis or equivalence partitioning exist that can be used in collaboration with combinatorial methods to systematically yield more efficient test sets.

C. Inference and Plan of Attack

As pointed out in the problem statement, it is impossible to provide the ego vehicle with the same inputs in each test run in regression testing. This is mainly due to the interactive character of the ego vehicle. Different behavior leads to different perspective views on the scenario space. Since the equality of the input space is therefore infeasible, we propose to strive for equality of the scenario space instead. Using dynamic scenarios, where dynamic objects adjust their motion online relative to the ego vehicle, makes a constant scenario space impossible. For that reason and, given the problems associated with the dynamic scenarios mentioned in Sec. III-B.1 and III-B.2, we opt for static and hybrid scenarios for regression testing of automated vehicles. Hereby the goal should always be to keep as much of the scenario static as possible. Only if otherwise not possible, dynamic parts should be introduced in order to obtain hybrid scenarios. In conclusion, the key in regression testing of autonomous vehicles is the independency of the scenario from dynamic parts. Static scenarios yield ego-vehicle-independent scenario space coverage, which makes them suitable for regression testing of autonomous vehicles. In addition, static scenarios allow for precise sequence coverage of events, which will be an essential part of the concept outlined in Fig. 1 and presented in more detail in Sec. V.

IV. SCENARIO DOMAIN PARAMETRIZATION

Our approach to find parameter values that in a second step can be combined with each other using CIT, follows the approach suggested in [4]. However especially when defining the motion of dynamic vehicles, we differ completely from the aforementioned publication to bypass the problems stated above. Instead of generating dynamic scenarios, we focus on generating static and hybrid scenarios. Below we give concrete examples of parameters that we use to generate test cases. Note that those parameters and their associated values are not a complete set.

A. Variation of the Road

Lane Marking Quality	Road Surface	
%	Type	%
0	Wet	0
25	Snow	25
50	Ice	50
75	Leaves	75
100		100

TABLE I

EXAMPLE: MUTABLE PARAMETERS OF THE ROAD SECTION.

Unlike [4] we are not varying the geometry of the base route. For simplicity we just consider one road section (for example, the merge example depicted in Fig. 2). Nevertheless, the quality of the road markings and the type of road surface are varied as shown in Table I.

B. Variation of Static Objects

Static Object		
x in m	y in m	Heading in °
10	550	10
11	551	15
12	552	20
13	553	25
14	554	30

TABLE II

EXAMPLE: PARAMETRIZATION FOR A STATIC OBJECT

After parameters that modify the road sections have been found, static objects are considered next. To achieve this, we combine parameters that describe the configuration of a static object. A configuration of a static object can be described entirely by its cartesian position and heading. Note that the geometry of the static object has to be defined as well.

C. Variation of Meteorological Effects

Sun/Moon			Clouds	Precipitation		Particles	
ϕ in °	θ in °	%	%	Type	%	Type	%
0	0	0	0	Rain	0	Fog	0
45	30	25	25	Snow	25	Dust	25
90	60	50	50	Hail	50		50
135	90	75	75		75		75
80		100	100		100		100
225							
270							
315							

TABLE III

EXAMPLE: METEOROLOGICAL EFFECTS EXPRESSED IN PARAMETERS THAT CAN INFLUENCE THE EGO VEHICLE

Since weather influences also play a big role in sensor measurements for autonomous vehicles, those parameters have to be varied as well [4]. As shown in Table III, we use sun, clouds and precipitation as well as particles in the air as main parameters to describe those effects. In more detail we describe the sun in spherical coordinates where ϕ represents the azimuth and θ the inclination of the sun. The distance r is not used as a parameter since it is proportional to the intensity of the sun. The intensity is a more meaningful parameter that can be varied between 0 and 100 percent. In the same way, the coverage of the sky, namely clouds, can be varied in the range between no clouds at all (clear sky) and fully covered. The type of cloud is neglected. To continue, the precipitation is a very important factor for testing autonomous vehicles. For example, depending on the type (rain, snow, hail) and intensity as a percentage, the camera system is influenced more or less.

D. Variation of Dynamic Objects

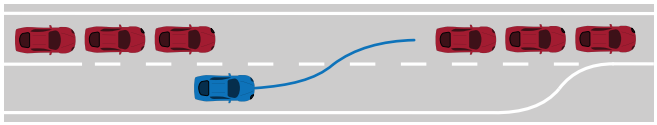

Fig. 2. Simple merge scenario: A group of *leading vehicles* and *following vehicles* in the continuing lane are forming a gap into which the ego vehicle (blue) is supposed to merge. The objective of the test is to evaluate, how the ego vehicle behaves for different behaviors of the red vehicles.

In contrast to the parameters covered so far, the parametrization of dynamic objects is not as straightforward as in the sections above. This is due to the fact that time as a parameter has to be taken into account. To illustrate this idea, we consider the example situation depicted in Fig. 2. The objective of this example situation is to evaluate how the ego vehicle (blue) merges into the gap between a group of leading and a group of following vehicles. The group of leading vehicles can be reduced to the last leading vehicle and the group of following vehicles can be reduced to the first following vehicle. This simplification can be made since the other vehicles of those groups are supposed to behave in the same way as those two gap-forming vehicles. At this point the question that must be answered is how many possible ways can the motion of those vehicles be combined with each other. To examine this more closely, the motion of the vehicles can be visualized in a path-time-diagram as depicted in Fig. 3.

We use a grid of points as a discretization of this path-time-space. By doing so, the motion of a dynamic object can be generated by connecting the points of the grid with each other. Hence, the points of the grid can be used as parameter values and therefore yield a possible parameterization for the dynamic objects as shown for one object in Table IV.

Building a covering array [11] using the parameters and parameter values as described above not only means that parameter values of the time sequence of each vehicle are combined with each other but also that all those values of different vehicles are combined with each other. To illustrate this idea, the following thought experiment can be considered. An error of the ego vehicle (SUT) might only occur if the leading vehicle is at position $s = 0m$ at $t = 0s$ and at position $s = 100m$ at $t = 8s$, while the following vehicle is required to be at position $s = 50m$ at $t = 7s$.

Of course the combinations must be constrained such that vehicles are not allowed to revert. If, for example, the s-value

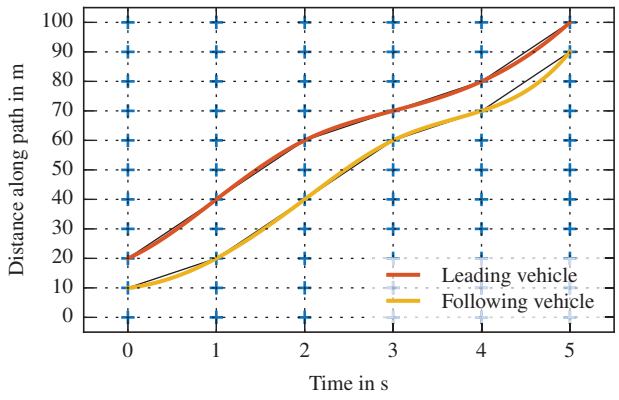

Fig. 3. Path-time diagram for the two representative vehicles that follow the same path and form a gap according to the example situation shown in Fig. 2. Observation: The motion of a vehicle along a predefined path can be defined by an interpolating spline passing through discrete points in the diagram.

Leading vehicle				Following vehicle			
t=0	t=1	t=2	t=3	t=0	t=1	t=2	t=3
s	s	s	s	s	s	s	s
0	0	0	0	0	0	0	0
10	10	10	10	10	10	10	10
20	20	20	20	20	20	20	20
30	30	30	30	30	30	30	30
40	40	40	40	40	40	40	40
50	50	50	50	50	50	50	50
60	60	60	60	60	60	60	60
70	70	70	70	70	70	70	70
80	80	80	80	80	80	80	80
90	90	90	90	90	90	90	90
100	100	100	100	100	100	100	100

TABLE IV

EXAMPLE: PARAMETERIZATION OF THE INPUT SPACE FOR TWO VEHICLES THAT FOLLOW THE SAME PATH FOR THE FIRST THREE SECONDS.

of the leading vehicle at $t = 0s$ is $30m$, then the value picked for $t = 1$ must be greater than that. Additionally, for each vehicle that is added to the scenario, we must ensure that the vehicles are not colliding with each other. Other constraints that have to be taken into account are of a dynamic/kinematic nature and are important because the goal is to generate motion that is as close to a human driver as possible. At large, the problem described above is a motion/trajectory planning problem that will be addressed in more detail below.

It should also be noted that the vehicles in the example above follow a predefined path, which reduces the complexity of the scenario space dramatically. In future this dimension should not be neglected, but instead be part of the motion planning and combinatorial testing algorithm.

E. Variation of the Ego Vehicle

According to Def. 6, the ego vehicle is also part of the scenario, which means that it can also be varied. Since the ego vehicle moves by its own logic, we only use the starting position and starting velocity as parameters. In the example depicted in Fig. 2, we even restrict the position to always be at the beginning of the merging window (first possible point of merge).

V. CONCEPT

The algorithm implemented to generate scenarios using the parameters obtained above, will be the topic of the remainder of this work. Therefore we devote a subsection to each block of the diagram depicted in Fig. 1. These blocks show the main components of the proposed algorithm. To simplify the explanation, we focus on the generation of motion for dynamic objects without the loss of generality for all the other parameters.

A. Combinatorial Interaction Testing (CIT)

Since the objective of this work is the generation of scenarios for regression testing, the combinatorial algorithm needs to be deterministic and instant (cf. [4]). Those requirements guarantee, that the scenario generation algorithm always generates the same test set if the provided input parameters remain the same. Moreover, the obtained test set is generated before any scenario is simulated and therefore can be saved to a database. Due to the fact that failures in complex software may only occur when more then just two parameters interact, the algorithm in addition needs to be able to generate t-way test sets. Constraint handling should also be supported since not all parameter value combinations express a valid input to the SUT. An algorithm that satisfies the aforementioned requirements is the greedy IPOG (In Parameter Order Generalized) algorithm as described in [6]. It is based on the IPO (In Parameter Order) algorithm and can be extended to handle constraints by using a list of forbidden tuples [7] or by solving a constraint-satisfaction problem using a constraint solver [8].

For the problem at hand, both approaches for constraint handling cannot be used. The forbidden tuples approach requires the constraints to be expressed as identities, which is not possible. Likewise it is not easily possible to reformulate the multi-vehicle motion-planning problem as a problem that can be solved by constraint programming, which is usually used for boolean, integer, linear or finite domains.

To solve the constraint-satisfaction problem, we use a modified version of the IPOG algorithm that uses a nonrecursive backtracking algorithm in conjunction with a trajectory planner as a feasibility checker. The IPOG algorithm has three logical parts where a validity check and an optimization are performed. When the permutations of t-tuples are generated, we check if all of those initial tuples represent feasible trajectories. Since reversing is not allowed, the arc length of subsequent parameters must be equal or higher than the previous one. Similarly we perform the second and third check on every horizontal and vertical extension of the array. This constraint already filters out many of the invalid combinations. In a second step, one has to check each t-tuple to see whether the combined arc lengths can be reached by a vehicle complying to kinematic and dynamic constraints. This evaluation requires a trajectory planner that is able to

plan a trajectory by connecting those parameter values with each other.

B. Backtracking

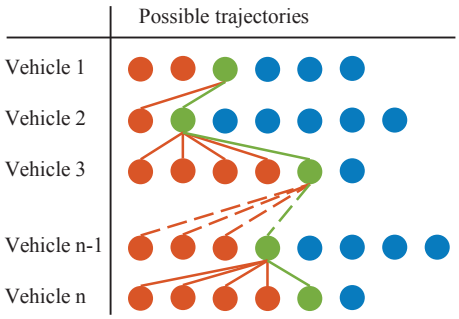

Fig. 4. Backtracking applied to the vehicle-trajectory-tree: Red: covered trajectory, Green: Current feasible trajectory, Blue: Uncovered trajectories, that have not been tested yet.

Backtracking algorithms can be used to solve constraint-satisfaction problems [16]. Trying to combine parameter values with each other using the IPOG approach yields such a problem. When trying to add a new path-time-tuple to an existing scenario, many possibilities exist for how to incorporate this tuple (cf. Sec. V-C). If the tuple, for example, is the first node that is supposed to define the trajectory of a vehicle, then the trajectory planner will connect this node to a list of starting and target nodes and thus yield many possible trajectories. The more tuples are added per vehicle, the more defined the trajectory becomes and the fewer possibilities exist.

In order to illustrate how we apply backtracking, the example depicted in Fig. 2 is used. If, in the horizontal extension of the covering array a new node (path-time tuple) that belongs to the following vehicle is supposed to be added, the trajectory planner that is associated with the following vehicle will use the current trajectory from the leading vehicle to impose constraints (cf. occupancy area in Fig. 5) on itself. Consequently, those constraint areas depend on the trajectory chosen from the first vehicle, which is also the reason we use backtracking. Of course a brute-force method could also be applied where the following vehicle simply would try to calculate a trajectory for every possible trajectory from the leading vehicle. With the backtracking approach, ideally only one trajectory is calculated for the leading vehicle, which is then used to generate the occupancy areas and to plan a trajectory with the new tuple for the following vehicle. If no trajectory can be found, the next trajectory for the leading vehicle is calculated, whereupon the following vehicle tries to plan a trajectory again. This going back and trying again is repeated until a solution is found. In the worst case, when no solution exists, all combinations have to be calculated, which yields the same complexity as the aforementioned brute-force approach. Of course the example above can be generalized for n vehicles as shown in Fig. 4. As soon as a solution for the $(n-1)$-vehicle has been found, the algorithm can proceed to find a solution for the n-th vehicle. For the case in which the parameter combination that ought to be added contains only parameters for some but not all objects and a solution for those parameters has been found, the backtracking algorithm needs to continue finding a solution for the remaining vehicles as well. This is important since subsequent vehicles are influenced by the new trajectories that have been generated according to the new parameter combination. Therefore, every time new parameters are added to the scenario, not only a local but a global solution needs to be found.

C. Motion Planning

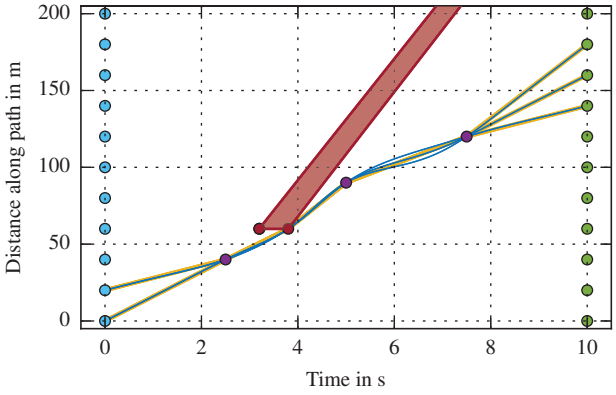

Fig. 5. Motion-planning example: First, the graph search yields connections (yellow) between start (cyan) and target nodes (green), that avoid the occupancy area (red) of another dynamic object, whereupon quadratic jerk minimal polynomials (blue) are interpolated.

As mentioned, a trajectory planner is necessary to plan trajectories according to input from the combinatorial testing algorithm. In robotics-related trajectory planners a certain start state is usually given. Using this state, the planner tries to find a trajectory to some terminating state that distinguishes the standard trajectory planning problem from the problem at hand. During horizontal or vertical extension of the covering array, the CIT algorithm samples points via the aforementioned backtracking algorithm in the path-time-space. Therefore, it is not guaranteed that the successively sampled points will start at $t = 0$ and will be strictly monotonic in time. It is possible that during the generation of the covering array, the trajectory of a vehicle is only defined by one path-time tuple, which has a time value different from the starting state and the target state. Thus, the planner needs to be slightly different from other robotics-related trajectory planners. The proposed planner works with a two-step procedure as described below.

1) Visibility Graph and Graph Search: In order to find a connection between two points in the path-time diagram depicted in Fig. 5, we use a visibility graph [9]. To be more precise, we first build a directed graph, that is then used to run graph search queries. We build the directed graph successively, node by node. Every time a new node is added to the graph, a connection to all already existing nodes is

established. For each connection, we then calculate the slope. If it surpasses certain limits, the edge is not added to the graph. For example, it is not possible to have a negative slope since that would mean that the vehicle is reversing. If the slope constraints are satisfied, an intersection check with the edges of the occupancy polygons is performed since those areas cannot be passed. Only if no intersection occurs do we calculate the euclidean distance between the two nodes in order to weight the new edge, which now can be added to the graph.

In order to plan a trajectory, the possible start nodes as well as end nodes, as depicted in Fig. 5, need to be defined. This is necessary since the combinatorial testing algorithm might first add a node or a combination of nodes between the start and target nodes. The planner then needs to be able to tell if a connection of one of the start nodes to one of the target nodes is possible while passing all CIT sampled in-between nodes. The occupancy polygons also need to be taken into account as demonstrated in Fig. 5. This also explains, why, before adding sampled nodes via CIT, the vertex nodes of the occupancy areas need to be added to the graph. To establish a connection from $t_{start} = 0s$ to t_{end}, start and end nodes are also added to the graph as described above. If the CIT algorithm now adds a new node to the graph via the backtracking algorithm, the planner creates a list of all possible connections that are the shortest from start nodes through the new nodes to target nodes. The shortest connections are calculated using Dijkstra's graph search, which makes use of the weighted edges calculated earlier. The piecewise shortest connections are then stitched together until this approach eventually yields a list of possible graph search connections. For each of those connections, a jerk minimal quintic polynomial needs to be interpolated as described below.

2) Jerk Minimal Quintic Polynomials: One of the required properties of the trajectory planner is the smoothness of the trajectory since the generated trajectories are supposed to imitate human driving behavior. Therefore we use quadratic jerk minimal quintic polynomials, as described in [10], to interpolate the possible connections of waypoints that have been found via the graph search. Of course those quintic polynomials need to be checked for intersections with the occupancy polygons as well. If a quintic polynomial passes this test in path-time-space, it is transformed into state-space, exploiting the differential flatness property of the nonlinear bicycle model that we used as a simplification of the dynamic behavior of the real vehicle, also described in [10]. This step is necessary since in state-space more constraints can be checked that determine if the trajectory is feasible (compare [10]).

VI. CONCLUSION

In this paper we tackled the problem of generating test cases/scenarios for regression testing of autonomous vehicles. By introducing some new terminology we aim to provide a foundation for future research and discussions regarding regression testing in this domain. Therefore we also pointed out difficulties that occur when researching this topic. As a partial solution to some of the stated problems, we proposed a new algorithm that can be used to generate static or hybrid scenarios. However, even though the algorithm works as expected, it needs to be optimized for more efficiency. In addition, it is only capable of creating motion along predefined paths. Future research should investigate the incorporation of the path-finding problem in the general concept which would allow for even more automation.

APPENDIX

A. VTD Driver Reacting to Ego Vehicle

Fig. 6. Gap opening up for the ego vehicle to merge in

The objective of the test case depicted in Fig. 6 is to test whether the ego vehicle is able to merge into the gap between the groups of leading and following vehicles. All vehicles are configured to be controlled by the VTD autonomous driver. The vehicles f02, f03 and f04, as well as l02, l03, l04 are supposed to follow their predecessor with a fixed distance (10m). The blue vehicle represents the SUT that, instead of speeding up and merging into the gap, tries to open up a new gap in the group of following vehicles. Those vehicles show cooperative behavior and make space for the ego vehicle.
VTD version: *2.0.2*

REFERENCES

[1] K. Bengler, K. Dietmayer, B. Farber, M. Maurer, C. Stiller, H. Winner, "Three Decades of Driver Assistance Systems: Review and Future Perspectives" *IEEE Intelligent Transportation Systems Magazine vol. 6*, 2014, pp. 622.

[2] J. E. Stellet, M. R. Zofka, J. Schumacher, T. Schamm, F. Niewels, J. M. Zllner, "Testing of Advanced Driver Assistance Towards Automated Driving: A Survey and Taxonomy on Existing Approaches and Open Questions," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, 2015, pp. 1455-1462.

[3] M. Maurer, J. C. Gerdes, B. Lenz and H. Winner, "The Release of Autonomous Vehicles" in *Autonomous driving: technical, legal and social aspects,* Springer Publishing Company, Incorporated, 2016.

[4] F. Schuldt, F. Saust, B. Lichte, M. Maurer and S. Scholz, "Effiziente systematische Testgenerierung für Fahrerassistenzsysteme in virtuellen Umgebungen," in *Automatisierungssysteme, Assistenzsysteme und Eingebettete Systeme Für Transportmittel*, 2013.

[5] S. Yoo, H. Mark, "Regression testing minimization, selection and prioritization: a survey," in *Software Testing, Verification and Reliability 22, no. 2*, 2012, pp. 67-120.

[6] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, J. Lawrence, "IPOG: A general strategy for t-way software testing," in *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, 2007, pp. 549-556.

[7] L. Yu, F. Duan, Y. Lei, R. N. Kacker, D. R. Kuhn, "Constraint handling in combinatorial test generation using forbidden tuples," in *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* 2015, pp. 1-9.

[8] L. Yu, Y. Lei, N. Nourozborazjany, R. N. Kacker, D. R. Kuhn, "An Efficient Algorithm for Constraint Handling in Combinatorial Test Generation," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation (ICST)*, 2013, pp. 242-251.

[9] T. Lazano-Perez, M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," in *Communications of the ACM Volume 22 Issue 10, Oct. 1979*, 1979, pp. 560-570.

[10] J. Schlechtriemen, K. P. Wabersich, K.-D. Kuhnert, "Wiggling through complex traffic: Planning trajectories constrained by predictions," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, 2016, pp. 1293-1300.

[11] D. R. Kuhn, R. N. Kacker, Y. Lei, "Combinatorial Methods in Testing," in *Introduction to Combinatorial Testing*, CRC Press, 2013.

[12] M. Dupuis, A. Biehn, "Preparation of scenarios," in *OpenDRIVE Scenario Editor - User Manual*, Vires Simulationstechnologie, 2016.

[13] M. Dupuis. (2017, May 29). *Vires* [Online]. Available: https://www.vires.com

[14] S. Geyer, M. Baltzer, B. Franz, et al., "Concept and development of a unified ontology for generating test and use-case catalogues for assisted and automated vehicle guidance," in *IET Intelligent Transport Systems*, 2014, pp. 183-189.

[15] S. Ulbrich, T. Menzel, A. Reschka, F. Schuldt, M. Maurer, "Defining and Substantiating the Terms Scene, Situation, and Scenario for Automated Driving," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems (ITSC)*, 2015, pp. 982-988.

[16] S. Russell, P. Norvig, "Solving Problems by Searching," in *Artificial intelligence: a modern approach, 3rd edition (December 11, 2009)*, Pearson, 1995.