Performance Comparison of Two Search-Based Testing Strategies for ADAS System Validation

Florian Klück, Martin Zimmermann, Franz Wotawa, Mihai Nica

Performance comparison of two search-based testing strategies for ADAS system validation

Florian Klück[1], Martin Zimmermann[1], Franz Wotawa[1], and Mihai Nica[2]

[1] Christian Doppler Laboratory for Quality Assurance
Methodologies for Autonomous Cyber-Physical Systems
Institute for Software Technology, Graz University of Technology,
Graz, Austria
{fklueck, mzimmerm, wotawa}@ist.tugraz.at
[2] AVL List GmbH, Graz, Austria
mihai.nica@avl.com

Abstract. In this paper, we compare the performance of a genetic algorithm for test parameter optimization with simulated annealing and random testing. Simulated annealing and genetic algorithm both represent search-based testing strategies. In the context of autonomous and automated driving, we apply these methods to iteratively optimize test parameters, to aim at obtaining critical scenarios that form the basis for virtual verification and validation of Advanced Driver Assistant System (ADAS). We consider a test scenario to be critical if the underlying parameter set causes a malfunction of the system equipped with the ADAS function (i.e., near-crash or crash of the vehicle). To assess the criticality of each test scenario we rely on time-to-collision (TTC), which is a well-known and often used time-based safety indicator for recognizing rear-end conflicts. For evaluating the performance of each testing strategy, we set up a simulation framework, where we automatically run simulations for each approach until a predefined minimal TTC threshold is reached or a maximal number of iterations has passed. The genetic algorithm-based approach showed the best performance by generating critical scenarios with the lowest number of required test executions, compared to random testing and simulated annealing.

Keywords: Autonomous vehicles · Genetic algorithm · Simulated annealing · System verification · Automatic testing.

1 Introduction

In order to improve functionality, increase user experience, or improve safety there is a trend of adding more and more automated and autonomous functions into our currently used devices and systems including cars. In case of the latter, Advanced Driver Assistant Systems (ADASs) have become more and more important in order to automate certain functions like keeping a lane on a highway or braking in case of an emergency, maybe finally leading to real autonomous

driving not requiring human drivers anymore. Obviously, it is of uttermost importance to keep systems safe especially in case of increased autonomy. Failing to provide an appropriate testing method results in severe accidents causing harm to people. One way of providing measures for keeping systems safe is to come up with a rigorous testing methodology that potentially identifies critical situations a system has to deal with. In this paper, we follow this research direction and provide an automated method for obtaining critical test scenarios.

In the context of our paper, a test scenario is a test case for an autonomous system comprising values for certain test parameters. These parameters are, for example, for setting the speed of the ego vehicle, i.e., the system under test, determining the number of pedestrians crossing a street or the behavior of any other entity that might interact with the system under test. Test scenarios can, therefore, be seen as instances of a general test scenario blueprint where we assign values for the parameters. The objective now is to find critical test scenarios, i.e., test scenarios that lead the system under test to crash (or at least to be close to such a situation). In order to find critical scenarios, we have to search for appropriate parameter values in an n-dimensional space where n is the number of parameters. Considering, in addition, that the parameters are often continuous numbers like the vehicle speed, there is a need for an efficient search procedure.

In this paper, we make use of two of such search algorithms, i.e., the genetic algorithm and simulated annealing, and compare them regarding their appropriateness for being used in the autonomous driving domain. We suggest an algorithm to be more appropriate if the algorithm requires fewer trials for finding a critical test scenario. A trial comprises setting the parameters and calling a simulation engine for executing the test scenario. Since simulation, requiring 3D simulation and physical simulation as well, is computationally demanding and time-consuming, limiting the number of necessary trials is, therefore, significant to make the approach practical. For making the comparison, we make use of the autonomous emergency braking system case study.

In contrast to our previous work [13], where we already introduced the general concept and the use of a genetic algorithm compared to random testing, we report on the following contributions in this paper:

- Introducing simulated annealing for finding parameters leading to a critical scenario in case of testing automated and autonomous systems.
- Comparing the performance of simulated annealing with random testing and our genetic algorithm approach using the autonomous emergency braking system case study.
- Introducing a novel heuristics function for evaluating the quality of tests during searching.

We organized the paper as follows: We first discuss the preliminaries comprising the tool-chain and the algorithmic foundations behind genetic algorithms and simulated annealing. Afterward, we compare the two search algorithms relying on previous research and introduce the underlying case study. The description of the case study comprises details of the setup, the obtained results, and a discussion. Finally, we discuss related research and conclude the paper.

2 Preliminaries

In this paper, we carry forward our previous work [13], where we contributed to automatic test case generation in an automotive context, by identifying critical scenarios that form the basis for virtual verification and validation of ADAS based driving functions. Again, we assume to have a set of parameters that comprise specific values in one particular driving scenario. We are interested in identifying specific values for parameter assignments that result in a driving scenario that can be classified as critical by means of our oracle function. In our case, the driving scenario can be seen as the input for testing and the classification regarding criticality as the output. The oracle function is explained in detail later in this paper.

Given a finite set of parameters P and a function dom that returns the domain, i.e., a set of values, for each parameter $p \in P$ as well as an oracle function Γ that returns *critical* $\bot$, in case the execution of the system under test (SUT) together with a given parameter value assignment leads to a critical situation, and *not critical* $\top$, otherwise. From a general perspective the main testing problem is based on the question, if there is an assignment of values PA from $dom(p)$ for all parameters $p \in P$ such that $\Gamma(\text{SUT}, PA)$ returns $\bot$, when executing the SUT. Notably, this testing problem may not be decidable for continuous parameter domains, since we are not able to exhaustively try all different parameter value combinations. For discrete parameter domains, the problem is decidable but it may still be infeasible to solve, due to the huge corresponding search space. In our previous work [13], we concluded that generally, at least for a simple testing problem, both genetic algorithm and random testing are able to find these value to parameter assignments that result in critical driving scenarios. However, evaluating every single execution is costly and time-consuming. Therefore, in this paper we want to investigate which testing strategy requires the least number of executions to find one suitable assignment of continuous values PA from $dom(p)$ for all parameters $p \in P$ such that $\Gamma(\text{SUT}, PA)$ returns $\bot$ at least once. We set up a case study to compare the performance of different search-based testing strategies in finding a suitable value to parameter assignment. As a reference, we compared the genetic algorithm and simulated annealing to random testing. All investigated testing strategies are explained in more detail in the upcoming chapters. The underlying tool-chain for automatic test scenario generation, execution and evaluation is based on previous work in this field [23, 14].

2.1 General Tool-chain Setup

To facilitate our rather comprehensive testing procedure, we designed the underlying tool-chain to automatically generate, execute and evaluate test scenarios, as pictured in Figure 1.

As an initial step, we select and prepare the testing strategy we want to apply and then use a script to configure and initiate the automatic testing procedure. A set of parameter values can be seen as a single test case that forms an executable test scenario when assigned to the appropriate parameters. Test cases

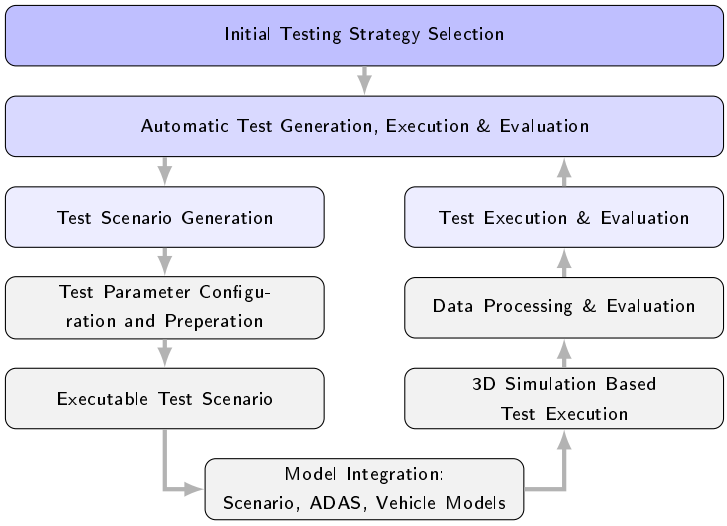


Fig. 1. Overview of the underlying tool chain for automatic test scenario generation, execution and evaluation.

are handed over one by one to a model integration and co-simulation platform called AVL Model.Connect [3]. This platform connects different models used for testing (e.g., vehicle models), driving functions as well as an interface to the 3D simulation platform Vires VTD [21]. Model.Connect generates a particular executable driving scenario, based on the obtained test case, and hands it over to Vires VTD for test execution. VTD exploits the input signals and gathers all required information about the virtual environment, for instance, sensor data. Next, all the data collected during test execution is sent back to Model.Connect, where we can access the gathered information via a script, to evaluate the driving scenario regarding criticality. Utilizing this framework allows us to not only carry out multiple experimental runs in sequence but also to execute and evaluate a theoretically unlimited number of test scenarios. Regarding time for testing, we enabled the simulation to run faster than real time and therefore reduced the total test duration by about one third.

2.2 Genetic Algorithm

Generally, genetic algorithms are based on the natural process of reproduction of an organism. The algorithm starts with an initial population and through reproduction and mutation, the algorithm evolves the initial population to a better, i.e., more optimal population. To solve an optimization problem, we can follow Algorithm 1. First, it generates a random start population, where the genes of each individual encode one variant of a solution to the optimization problem. Each individual is then evaluated based on a cost function. Until the number of desired generations is reached, the best individuals are selected from

the current population and will be crossed with each other to generate new individuals. In addition to crossing, a mutation of single genes might happen, where random changes to the genes occur with a certain probability. Finally, the new population is evaluated again [9].

Algorithm 1 Basic genetic algorithm

Require: A maximum number of generations G, which should be produced, a method to generate a seed population *generate*, a method to select a new population from an old population *select*(P), a method to evaluate the population *evaluate*(P), a method to cross the population *cross*(P) and a method to mutate a population *mutate*(P).

```
1: procedure GeneticAlgorithm(G, generate, select, evaluate, cross, mutate)
2:     P ← generate()
3:     evaluate(P)
4:     for g = 0 to G do
5:         P ← select(P)
6:         P ← cross(P)
7:         P ← mutate(P)
8:         evaluate(P)
9:     end for
10: end procedure
```

The biggest challenge when using a genetic algorithm is to find a suitable encoding for the genes and to design appropriate functions that accurately guide the key processes of generation, selection, crossing and mutation. For our case study, we are using a modified version of the DEAP Python library [7]. The main modification applied for this paper solely stops the GA right after our target value is reached, i.e., this means stopping the GA in the middle of evaluating a generation. As mentioned above, selecting the right encoding for the genes and defining appropriate functions is not trivial and often can only be done by trial and error. We decided to encode the parameters for our case study directly as values from $\mathbb{R}$ in the genes. Our initial population consists of 100 randomly generated individuals. To select individuals for the next generation we use tournament selection with a tournament size of 3. As a crossing function, we are using uniform crossing. After the crossing, each parameter of an individual has a 50% chance to be mutated in the following way. First, we choose a random value between $-\Delta p_i$ and Δp_i, where Δp_i is a value defined per parameter, which indicates how much the mutation is allowed to deviate from the current value. This random value is then added to the current parameter value. The evaluation function will be described in detail in Section 4.

2.3 Simulated Annealing

Simulated annealing and genetic algorithm are two distinct optimization techniques that share one particular commonality: they are both inspired by natural

processes. While a genetic algorithm models the process of natural selection, simulated annealing is based on the metallurgic challenge to heat and cool a metal properly during processing so that an optimal crystal structure forms. Metals with high temperature and therefore high thermodynamic energy show a loose and very flexible crystal structure that becomes more and more rigid with decreasing temperature. However, if the metal is rapidly quenched, the formed crystal structure is uneven and the metal becomes brittle. During the cooling process, however, to change from an initial sub-optimal crystal structure to an optimal structure, sometimes an intermediate state is required that is even worse, compared to the initial crystal structure. Therefore, simulated annealing also considers bad solutions on a transitional basis, when approximating the global optimum for a given function. The general process behind simulated annealing can be described by the help of two main parameters: temperature and energy. Objects in nature commonly pursue a state with the lowest level of energy possible. Transferred to simulated annealing, a low level of energy represents a good solution. Therefore, the initial solution is modified until a new better solution is found that results in a lower level of energy, compared to the initial solution. Based on the temperature, a probability for intermediately accepting a worse solution is defined. As long as the temperature is high, worse solutions are more likely to be accepted as intermediate states in order to maintain the possibility to leave a local optimum on the way to the global optimum. However, the temperature parameter decreases over time, therefore the chance to leave local optima decreases over time as well. The general working principle for simulated annealing is illustrated in Algorithm 2.

Algorithm 2 Basic simulated annealing algorithm

Require: A maximum number of generated solutions S, a method to generate a starting solution *generate*, a method to evaluate a solution *evaluate*(s), a method that lowers the temperature $T(t)$, a method to pick a new solution from an existing one *move*(s) and an acceptance function $P(e', e, t)$.

```
1: procedure SimulatedAnnealing(S, generate, move, evaluate, P)
2:     s ← generate()
3:     e ← evaluate(s)
4:     for i = 0 to S do
5:         t ← T(t)
6:         s' ← move(s)        ▷ Is changed to s' ← move(s, t) in Locatelli's algorithm
7:         e' ← evaluate(s')
8:         if P(e', e, t) ≥ random(0, 1) then
9:             s ← s'
10:            e ← e'
11:        end if
12:    end for
13: end procedure
```

In our case study, we first used a standard version of the simulated annealing algorithm (SA), where the move function behaves the same as the mutation function of the genetic algorithm. Second, we implemented an improvement suggested by Locatelli [16] (Loc. SA) where a sphere around the point, from which new solutions are drawn, shrinks with decreasing temperature. In our case, the sphere covers the whole parameter range at maximum temperature and shrinks proportional until it collapses to a point at the minimum temperature. For both algorithms, we used the Python library Simanneal [18] as a base and modified it accordingly, to include Locatelli's algorithm. We decrease the temperature linearly and the acceptance function is the standard simulated annealing acceptance function depicted in Equation 1, where e' is the evaluation of the new solution, e is the evaluation of the old solution, and t is the current temperature. Hence, the acceptance function always accepts if the new solution is better or equal to the old solution. If not, a value between 0 and 1 is returned depending on the temperature and the difference between the evaluations of the two solutions. When the system still has a high temperature, the probability for accepting a worse solution will be higher. Similar if the evaluations of the two solutions are close together, the probability for accepting a worse solution will be higher, and if the evaluations of the solutions are far apart, the probability will be smaller. As for the genetic algorithm, we will describe the evaluation function in Section 4

$$P(e', e, t) \begin{cases} 1, & \text{if } e \leq e' \\ \exp(\frac{-(e'-e)}{t}), & \text{otherwise} \end{cases} \tag{1}$$

3 Comparison of Genetic Algorithm and Simulated Annealing

As introduced in the previous sections genetic algorithm and simulated annealing represent two search techniques that utilize heuristic meta information to find an optimal solution to a problem. In most applications, these algorithms are used to solve combinatorial optimization problems, e.g., the traveling salesman problem. However, they have also successfully been applied to continuous optimization problems as described in [6]. In our case study, we want to follow this path of research and apply simulated annealing to a continuous optimization problem in the context of automated and autonomous systems. Furthermore, we want to investigate the performance of simulated annealing by comparing it to the genetic algorithm as done by [17]. Here, Manikas and Cain compared the performance of simulated annealing and genetic algorithm on a combinatorial optimization problem, namely the circuit partitioning problem. As a conclusion they stated that the genetic algorithm is probably more suitable than simulated annealing when approximating the optimal solution to a function comprising discrete values. However, they did not provide much information on how many individual solutions they considered to get to these results, or in other words, how long it took the two algorithms to come up with the optimal solutions.

Regarding execution time, a better indication is provided by [1], where Akinwale et al. compared simulated annealing and the genetic algorithm by applying both search techniques on the timetabling problem, representing another combinatorial problem. They stated that both algorithms produce feasible solutions, however, the execution time for simulated annealing to find such a solution was found to be 2.5 times higher than for the genetic algorithm. Hereafter, Fredrikson and Dahl further investigated the timetabling problem and concluded that the genetic algorithm is faster in the beginning while simulated annealing performs better in the later stages [8]. While most related work in this direction concludes that a genetic algorithm is probably better suited for approximating the optimal solution to a problem, we could not find any work that compares the performance of both algorithms regarding execution time in detail. Therefore, our case study will follow this path of research and analyze the required execution time of both algorithms to find a certain threshold value, representing a good solution to a continuous optimization problem in the context of automated driving based on Advanced Driver Assistant Systems.

4 Case Study

In the following, we report on the results obtained from the case study. First, we describe the underlying experimental setup, followed by presenting and discussing the obtained empirical results.

4.1 Setup

The case study described in this paper is a continuation based on previous research [13], where we investigated the effectiveness of our genetic algorithm approach in finding critical scenarios by comparing it to results obtained when using random testing. Here, we demonstrated that the genetic algorithm reliably finds parameter value combinations that lead to critical driving scenarios. Furthermore, we showed that invalid test scenarios are iteratively eliminated, when the genetic algorithm modifies the generation of new scenarios accordingly. However, random testing did also produce suitable results, which led to the assumption that the genetic algorithm might perform better on driving scenarios with higher complexity, containing more traffic participants, various driving maneuvers and in general more underlying parameters to control.

For the current case study, we made several extensions to the driving scenario in order to increase the complexity. The new driving scenario covers 16 different parameter domains and describes the ego vehicle (EGO) driving behind another vehicle, the global vehicle target (GVT), on a straight road with a cluster of two parking vehicles on the right side as shown in Figure 2. In addition, two pedestrians are moving either aside or across the road. The first pedestrian can cross the street in various angles from the left, the second pedestrian can walk across the street either behind the second car, between the two cars, or in front of the first car. In total we have 16 different parameter domains that act as

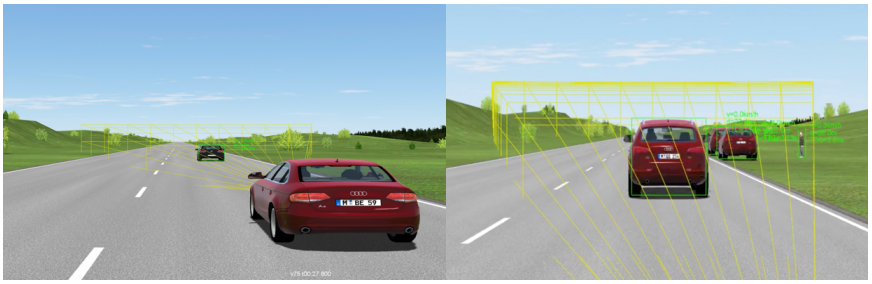


Fig. 2. This exemplary simulation run shows the Ego vehicle (following vehicle), the GVT vehicle (leading vehicle) as well as a cluster of parked vehicles aside the road with one pedestrian ready to cross the street. Within the sensor range (yellow cone) green bounding boxes are drawn around detected objects.

placeholders in the predefined scenario template, forming an executable driving scenario as soon as continuous parameter values are assigned accordingly. This executable driving scenario is then simulated for 90 seconds in our tool-chain as described in Section 2.1. Hereby, the genetic algorithm approach iteratively optimizes test parameter values from a given seed population. Simulated annealing modifies parameter values by selecting new values within a variable range embracing the current parameter values. For random testing an arbitrary set of parameter values is selected to form an executable scenario. The resulting driving scenarios form the basis for virtually testing the ego vehicle equipped with an AEB system. The AEB system is a vehicle active safety function, designed to automatically trigger a braking maneuver in response to the detection of a potential collision. The system comprises two main modules, first the vision system and second the brake control system. The vision system (I) is based on a perfect radar sensor model, enabling the system to detect even covered objects, as well as an object detection algorithm. As soon as an object enters the radar's cone-shaped field of view (i.e., within 250 meters), the system detects the object and identifies its position and speed. Based on that, the brake control system (II) calculates the minimum required braking distance towards the object and adds a velocity dependent safety margin. As soon as the calculated distance is equal to or goes below the actual headway distance, an appropriate braking maneuver is triggered in accordance with the situation.

The main objective of the case study, presented in this paper, is to investigate how long it takes, for each of the three different approaches, to find a satisfiable solution (i.e., a critical driving scenario). In other words, for each approach, we want to investigate how many test executions are required to generate a solution that meets the predefined evaluation target. Regarding evaluation, we rely on time-to-collision (TTC), a well-known and commonly used indicator to identify potential rear-end conflicts [12]. According to previously conducted experiments, based on the virtual driving scenario we consider in this case study, we found that a TTC below 0.3 seconds represents the transition area from a near-crash

scenario to an actual crash scenario. Therefore, we set the target value for our evaluation criteria to 0.3 seconds, i.e., the oracle function Γ returns *critical* $\perp$ in this case. Furthermore, based on previous work as described in [13], we made several improvements to the evaluation function itself, to guide the generation of a critical scenario more efficiently. Therefore, we introduced five different criticality zones that we monitor throughout the simulation and use the obtained information as additional input for our evaluation function. The zones are listed from least critical to most critical.

1. No object is within a 250m radius of the EGO
2. An object is anywhere within a 250m radius of the EGO
3. An object is in the lane of the EGO and less than 250m away
4. An object is in front of the EGO produces a TTC of less than 20s
5. An object is in front of the EGO produces a TTC of less than 0.3s

The genetic algorithm and simulated annealing are designed to modify parameter assignments such that the resulting driving scenario traverses from low criticality zones to higher criticality zones over time. We automatically executed each method seven times to compare how many test executions are required in average and median to find a driving scenario for which the oracle function Γ returns *critical* $\perp$ (i.e., TTC below 0.3 seconds), before a maximum number of 500 iterations has passed.

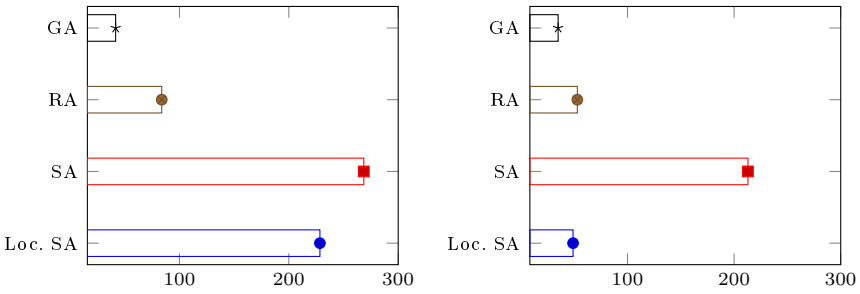
4.2 Results

As described in the previous section, we carried out further experiments based on the AEB case study, described in [13], to investigate how many test executions are required in order to find a suitable solution (i.e., a driving scenario for which the oracle function Γ returns *critical* $\perp$), when applying genetic algorithm, simulated annealing and random parameter selection. Therefore, we executed each method seven times until either a resulting TTC of 0.3 seconds was detected or a maximum number of 500 iterations has passed. Clearly, we are hereby searching for a solution that meets a certain threshold value, rather than searching for the global optimum. In the following, we want to report on the empirical results obtained from our experiments, as summarized in Table 1.

Table 1. Overview on the test results obtained from seven experimental runs for every investigated test methods.

	Average	Minimum	1. Quartile	Median	3. Quartile	Maximum
GA	41.71	18	19	35	65	70
RA	83.85	4	20	53	127	235
SA	268.57	40	131	213	500	500
Loc. SA	228.42	6	20	49	500	500

As the first column of Table 1 shows, overall it took the genetic algorithm (GA) the lowest number of test executions, in average 42 executions, to generate a test scenario that results in a TTC value below 0.3 seconds. In addition, the genetic algorithm showed very consistent performance and reliably found a suitable solution in each of the seven experimental runs. In comparison, the second row of Table 1 summarizes the results obtained when applying random parameter value selection (RA). For random testing in average 84 executions are required to generate a test scenario that results in a TTC value below 0.3 seconds. Also for random testing a suitable solution (i.e., a driving scenario for which the oracle function Γ returns *critical* $\perp$) was found in each of the seven test runs. The highest number of executions is on average required for both simulated annealing approaches, where it took the basic algorithm (SA) on average 269 executions and the algorithm proposed by Locatelli (Loc. SA) on average 229 executions, to generate a suitable solution. The high number of test executions for Loc. SA in the 3. Quartile in contrast to low median shows that it either finds a suitable solution really fast or not at all (i.e., termination after 500 iterations), which can be explained by the general working principle of the Loc. SA. Not finding a solution at all might happen when the initial solution, which the Loc. SA randomly generates, is very far away from the global optimum and it also does not find a good solution within the first few iterations. Both simulated annealing algorithms have in common that when searching for the global optimum, the algorithms are very likely to get stuck in one of the several local optima, which are better than the initial solution. With decreasing temperature over time, also the possibility of leaving such a better suited but still insufficient local optimum is decreasing. The advantage of the simulated annealing algorithm proposed by Locatelli, compared to the basic SA, is that the algorithm is able to take significantly greater steps through the search space, depending on the temperature. This advantage becomes clearly visible if we compare the average and the median number of required executions, as visualized for all investigated test methods in Figure 3.



(a) Average number of executions (b) Median number of executions

Fig. 3. Comparison of (a) average and (b) median number of required execution per test method.

Here it can be seen that for genetic algorithm (GA), random testing (RA) as well as for the basic simulated annealing algorithm (SA), the difference between average and median number of required executions is comparatively small. However, for the simulated annealing algorithm as proposed by Locatelli (Loc. SA), we observe a major gap with an average number of 229 and a median number of 49 required executions. Considering every individual result obtained from the seven Loc. SA runs, we observe that if the Loc. SA has not found a suitable solution after 49 executions, it will not find a solution at all but run into the stop criterion instead, which is triggered after 500 test executions. This observation is reasonable due to the fact that for the Loc. SA algorithm the temperature parameter is proportionally decreasing with every execution towards a minimal temperature value, which is reached after 500 executions. In other words, with an increasing number of test executions, the sphere that defines the search space around the current solution is steadily tightening until no new solutions are produced at all. In contrast, the basic SA works with a fixed search space and could technically still accept better solutions after 500 iterations, eventually finding a suitable solution if an arbitrary number of test executions would be permissible. It would be interesting to see if the performance of the Loc. SA can be further increased if we define an individual stop criterion that restarts the algorithm, if no suitable solution is found after a certain number of test executions, indicating an unsuitable initial solution. We will follow this approach as part of our future research. The summarized results in Table 1 are visualized in Figure 4 to be further investigated and to be compared in more detail.

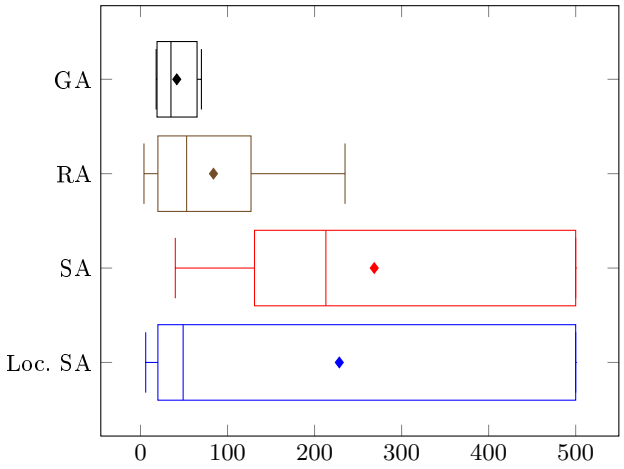


Fig. 4. Comparison of Box-Plots comprising the results obtained from seven test run for each investigated method

Here again, we see that the genetic algorithm finds suitable solutions in the fastest and most effective manner. The corresponding box plot summarizes the

distribution of results obtained from all seven experimental runs. It can be observed that the minimum number (vertical line on the left) of 18 required executions and the maximum number (vertical line on the right) of 70 required executions closely embrace the interquartile range (box), showing that 50 percent of all test runs required between 19 and 65 executions until a suitable solution was found. Notably, the median number (vertical line in box) of 35 required executions does not deviate much from the average number (diamond in box) of 42 required executions. This leads to the interpretation that the genetic algorithm performs very consistently and produces results that are not much affected by any outliers.

For random testing, it can be observed that 50 percent of all test runs required between 20 and 127 executions until a solution was found, for which the oracle function Γ returns *critical* $\perp$. However, random testing is slightly stronger affected by outlier results in both directions compared to the genetic algorithm, as indicated by a minimum number of 4 required executions and a maximum number of 235 required executions. This observation is also supported by a bigger deviation between the average and the median number of required executions. Based on our results, the basic simulated annealing algorithm performed worse, compared to the other investigated methods. Here, on average 269 executions were required to find a solution. 50 percent of all test runs required more than 213 executions to find a suitable solution, as indicated by the median, or terminated before finding any good solution at all. The deviation between average and median required number of executions is greater than for the genetic algorithm and for random testing, indicating that the basic simulated annealing algorithm is strongly affected by outlier results.

Finally, for the simulated annealing algorithm as proposed by Locatelli we can observe that the box plot more or less covers the whole area of possible results. Notably, 50 percent of all test runs required between 6 and 49 executions to find a suitable solution, which is comparable to the performance we observed for the genetic algorithm and random testing. However, in this case, the deviation between average and median required number of executions is significantly greater than for all the other investigated methods. This underlines our previously described assumption, that also the Loc. SA either finds a suitable solution very fast or not at all, depending on the quality of the initial solution. As previously mentioned, we find this observation quite interesting and will make further adjustments to the Loc. SA as part of our future work. Hereby, we plan to introduce a new internal stopping criterion that triggers the generation of a new initial solution, if necessary.

4.3 Discussion

In the subsection Results, we discussed the experimental results obtained from the case study, where we investigated which testing strategy requires the lowest number of executions to generate a critical scenario (i.e., resulting in TTC values below 0.3 seconds). In the course of the case study, we compared the performance of a genetic algorithm to simulated annealing and random testing.

Hereby, the genetic algorithm showed the best performance in finding the desired value to parameter assignments that lead to a critical driving scenario, with regards to the required number of test executions. Furthermore, for the basic simulated annealing algorithm as well as for the simulated annealing algorithm proposed by Locatelli, both strength and weaknesses were detected. On the one hand, depending on the quality of the initial solution, both simulated annealing algorithms are generally able to find a suitable solution within a low number of test executions. However, on the other hand, if the initial solution is of bad quality, both simulated annealing algorithms are very likely to get stuck in local optima rather than generating a critical driving scenario. Although the observed data is very interesting, we acknowledged that the results of only 7 test runs per method do not represent a sufficiently large data set to conduct meaningful statistical analysis. However, we consider the presented evaluation as an initial basis for conducting larger experiments in our further work. Also based on the observation that Loc. SA performance either good or bad, we plan to make further improvements to the simulated annealing algorithm proposed by Locatelli as part of our future research.

5 Related Work

One of the biggest challenges for testing ADAS systems is caused by the near infinite input domain. Due to that, Rafaelli proposed that stochastic approaches should be preferably used for ADAS systems verification and validation, rather than deterministic approaches and test protocols [19]. Therefore, the idea to use stochastic and search-based testing methods for system testing is not novel. In their paper Gross et al. compared search-based testing on module or unit level to search-based testing on system level. They concluded that search-based testing is only meaningful when it is performed on a system level. Otherwise, a large number of false positive test cases is generated that only reveal the execution of an invalid input sequence. However, when search-based testing methods are used on system level all revealed faults have been shown to be actual faults of the system [10].

In the automotive direction, there are also a few papers available that describe frameworks for testing ADAS functions [15, 20, 26, 25]. However, none of these frameworks provide an automated method for ADAS system testing. In literature we can also find some papers on parameter optimization utilizing a genetic algorithm [24, 22], as well as utilizing a simulated annealing algorithm [11]. In these papers, both methods are usually described to perform better or comparably good compared to a random selection of parameter values. Furthermore, genetic algorithms have also been used in software testing, for example, to find tests that cause extreme processing times as described in [2].

Moreover, a few studies have also been conducted on testing ADAS functions by utilizing a genetic algorithm or similar search-based algorithms. Ben Abdessalem et al. have created their own algorithm based on a genetic algorithm and applied it for testing a pedestrian detection system [4]. Similar to

our paper, [5] compared random testing with evolutionary test case generation. However, they left a few research questions unanswered, which we took into account in our last paper [13], where we also focused on testing an AEB function and compared random test case generation to test case generation guided by a genetic algorithm.

6 Conclusion

In this paper, we have compared the use of a genetic algorithm and simulated annealing for test parameter optimization in the context of autonomous and automated driving. Especially in case of advanced driver assistant systems, it is of uttermost importance to find instances of usage scenarios that might lead to crash situations in order to verify the implemented functionality. Because of the limited amount of resources it is, therefore, important to introduce an automated method. For this purpose we suggest a combination of search-based testing and system simulation. The goal of this paper is to evaluate two algorithms for implementing search-based testing using the autonomous emergency braking system. In addition to previous research, we also modified and improved the heuristics function for evaluating solutions.

The results of the empirical evaluation showed that the genetic algorithm is superior compared to simulated annealing and also random testing. Using a genetic algorithm for test parameter optimization for the autonomous emergency braking system requires fewer test executions on average and on median compared to the other methods. In future research, besides improving the Loc. SA, we will further investigate on this comparison using more and larger case studies from the autonomous driving domain.

Acknowledgment

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged. We also want to thank our colleague Jianbo Tao for supporting the tool-chain set-up and AVL List GmbH for providing the underlying simulation models and software tools required for the empirical evaluation. The investigated prototypical AEB function was developed in the course of internal research activities and is solely used for demonstration purpose.

References

1. Akinwale, C., Olatunde, S., Olusayo, E., Babalola, J., et al.: Performance evaluation of simulated annealing and genetic algorithm in solving examination timetabling problem. Scientific Research and Essays **7**(17), 1727–1733 (2012)
2. Alander, J.T., Mantere, T., Turunen, P.: Genetic algorithm based software testing. In: Artificial Neural Nets and Genetic Algorithms. pp. 325–328. Springer Vienna, Vienna (1998)

3. AVL List GmbH: Model.connect, https://www.avl.com/-/model-connect-
4. Ben Abdessalem, R., Nejati, S., Briand, L.C., Stifter, T.: Testing advanced driver assistance systems using multi-objective search and neural networks. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. pp. 63–74. ASE 2016, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2970276.2970311, http://doi.acm.org/10.1145/2970276.2970311
5. Buehler, O., Wegener, J.: Evolutionary functional testing of a vehicle brake assistant system. In: 6th Metaheuristics International Conference, Vienna, Austria. Citeseer (2005)
6. Erdogmus, P., Ozturk, A., Tosun, S.: Continuous optimization problem solution with simulated annealing and genetic algorithms. Journal of Engineering Research and Applied Science **2**(1), 116–121 (2013)
7. Fortin, F.A., De Rainville, F.M., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research **13**, 2171–2175 (jul 2012)
8. Fredrikson, R., Dahl, J.: A comparative study between a simulated annealing and a genetic algorithm for solving a university timetabling problem (2016)
9. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edn. (1989)
10. Gross, F., Fraser, G., Zeller, A.: Search-based system testing: High coverage, no false alarms. In: Proceedings of the 2012 International Symposium on Software Testing and Analysis. pp. 67–77. ISSTA 2012, ACM, New York, NY, USA (2012). https://doi.org/10.1145/2338965.2336762, http://doi.acm.org/10.1145/2338965.2336762
11. Haddock, J., Mittenthal, J.: Simulation optimization using simulated annealing. Computers & Industrial Engineering **22**(4), 387 – 395 (1992). https://doi.org/https://doi.org/10.1016/0360-8352(92)90014-B, http://www.sciencedirect.com/science/article/pii/036083529290014B
12. Kircher, K.: A comparison of headway and time to collision as safety indicators. Accident; analysis and prevention **35**, 427–33 (06 2003). https://doi.org/10.1016/S0001-4575(02)00022-2
13. Klück, F., Zimmermann, M., Wotawa, F., Nica, M.: Genetic algorithm-based test parameter optimization for adas system testing. In: Proceedings of the 19th International Conference on Software Quality, Reliability and Security : (2019)
14. Klueck, F., Li, Y., Nica, M., Tao, J., Wotawa, F.: Using ontologies for test suites generation for automated and autonomous driving functions. In: 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). pp. 118–123 (Oct 2018). https://doi.org/10.1109/ISSREW.2018.00-20
15. Lattarulo, R., PÃ©rez, J., Dendaluce, M.: A complete framework for developing and testing automated driving controllers. IFAC-PapersOnLine **50**(1), 258 – 263 (2017). https://doi.org/https://doi.org/10.1016/j.ifacol.2017.08.043, http://www.sciencedirect.com/science/article/pii/S2405896317300587, 20th IFAC World Congress
16. Locatelli, M.: Simulated annealing algorithms for continuous global optimization: Convergence conditions. Journal of Optimization Theory and Applications **104**(1), 121–133 (Jan 2000). https://doi.org/10.1023/A:1004680806815, https://doi.org/10.1023/A:1004680806815

17. Manikas, T.W., Cain, J.T.: Genetic algorithms vs. simulated annealing: A comparison of approaches for solving the circuit partitioning problem. Computer Science and Engineering Research. 1. (1996)
18. Perry, M.: simanneal: Python module for simulated annealing. https://github.com/perrygeo/simanneal (2013–2019)
19. Raffaelli, L., Vallée, F., Fayolle, G., De Souza, P., Rouah, X., Pfeiffer, M., Géronimi, S., Pétrot, F., Ahiad, S.: Facing ADAS validation complexity with usage oriented testing. In: ERTS 2016. p. 13. Toulouse, France (Jan 2016), https://hal.inria.fr/hal-01277494
20. Semrau, M., Erdmann, J.: Simulation framework for testing adas in chinese traffic situations. SUMO 2016–Traffic, Mobility, and Logistics **30**, 103–115 (2016)
21. VIRES Simulationstechnologie GmbH: Vtd – virtual test drive, https://vires.com/vtd-vires-virtual-test-drive/
22. Wang, Q.: Using genetic algorithms to optimise model parameters. Environmental Modelling & Software **12**(1), 27 – 34 (1997). https://doi.org/https://doi.org/10.1016/S1364-8152(96)00030-8, http://www.sciencedirect.com/science/article/pii/S1364815296000308
23. Wotawa, F., Peischl, B., Klück, F., Nica, M.: Quality assurance methodologies for automated driving. e & i Elektrotechnik und Informationstechnik **135**(4), 322–327 (Aug 2018). https://doi.org/10.1007/s00502-018-0630-7, https://doi.org/10.1007/s00502-018-0630-7
24. Wright, A.H.: Genetic algorithms for real parameter optimization. Foundations of Genetic Algorithms, vol. 1, pp. 205 – 218. Elsevier (1991). https://doi.org/https://doi.org/10.1016/B978-0-08-050684-5.50016-1, http://www.sciencedirect.com/science/article/pii/B9780080506845500161
25. Zhou, J., Schmied, R., Sandalek, A., Kokal, H., del Re, L.: A framework for virtual testing of adas (apr 2016). https://doi.org/https://doi.org/10.4271/2016-01-0049, https://doi.org/10.4271/2016-01-0049
26. Zofka, M.R., Klemm, S., Kuhnt, F., Schamm, T., Zöllner, J.M.: Testing and validating high level components for automated driving: simulation framework for traffic scenarios. In: 2016 IEEE Intelligent Vehicles Symposium (IV). pp. 144–150 (June 2016). https://doi.org/10.1109/IVS.2016.7535378