Models and algorithms for the exploration of the space of scenarios : toward the validation of the autonomous vehicle

Marc Nabhan

université PARIS-SACLAY

Models and algorithms for the exploration of the space of scenarios: toward the validation of the autonomous vehicle

Thèse de doctorat de l'Université Paris-Saclay

École doctorale n°580, Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat: Informatique
Unité de recherche: Université Paris-Saclay, Inria, Inria Saclay-Île-de-France, 91120, Palaiseau, France
Référent: Faculté des sciences d'Orsay

Thèse présentée et soutenue en visioconférence totale, le 21 décembre 2020, par

Marc NABHAN

Composition du Jury

Sébastien VEREL Professeur, Université du Littoral Côte d'Opale	Président
Jin-Kao HAO Professeur, Université d'Angers	Rapporteur & Examinateur
Antoine GRALL Professeur, Université de Technologie de Troyes	Rapporteur & Examinateur
Dominique QUADRI Professeure, Université Paris-Saclay	Examinatrice
Marc SCHOENAUER Directeur de recherche, Inria Saclay	Directeur de thèse
Yves TOURBIER Ingénieur de recherche, Renault	Co-Encadrant & Examinateur
Hiba HAGE Ingénieure de recherche, Renault	Co-Encadrante

Thèse de doctorat

NNT: 2020UPASG058

Synthèse

Les véhicules autonomes et les aides à la conduite automobile représentent le futur du transport routier. Hypothétiquement, ils sont supposés augmenter globalement la sécurité routière. Cependant, ce sont des systèmes assez complexes, de la coordination de multiples capteurs à leur fusion, jusqu'à la loi de commande qui décide des actions que le véhicule exécutera. Une défaillance peut se produire durant n'importe quel stade de ce processus, ce qui va enclencher une fausse action sur la route. C'est pourquoi chaque composant de ce système devra être rigoureusement testé pour anticiper des défaillances potentielles et les éliminer.

Les essais de conduite réelle sont utilisés pour tester le véhicule sous diverses conditions et identifier des défaillances spécifiques. Cependant, cette approche ne peut pas être utilisée seule pour compléter le processus de validation, en sachant que plusieurs études ont montré qu'il faudrait des siècles de conduite continue pour démontrer que le taux de défaillance du véhicule autonome est inférieur à celui du conducteur humain. C'est pourquoi, grâce à l'essor des puissances de calcul, des méthodes de test par simulation sont utilisées pour complémenter les essais réels, et sont nettement moins chères. Une de ces méthodes est la simulation numérique, qui génère des données virtuelles paramétrées et recherche de nouvelles situations à travers un simulateur de conduite.

Le contexte de cette thèse s'inscrit dans la simulation numérique. Son objectif est de faciliter la validation de la loi de commande par des tests en MIL (Model-In-the-Loop) où l'environnement de la loi de commande est simulé sans composants physiques. Nous pouvons donc vérifier que les actions choisies par le véhicule demeurent en sécurité selon les règles de conduite, et valider numériquement les exigences de la loi de commande. Pour cela, nous explorons une multitude de scénarios, qui sont des combinaisons de paramètres d'entrée qui définissent la situation de conduite appelée aussi "use case", à travers le simulateur SCANeR Studio.

Cette thèse CIFRE fait partie d'un projet industriel à Renault appelé ADValue. Son objectif est de combiner plusieurs algorithmes mis en compétition pour explorer efficacement l'espace des paramètres d'entrée d'un "use case" afin d'identifier toutes les zones défaillantes. Pour cela, divers modèles et algorithmes sont développés pour exploiter les ressources disponibles d'une manière efficace afin d'éviter de simuler toutes les conditions imaginables pour tous les "use cases", ce qui est intraitable en puissances de calcul. Le but de cette thèse est de contribuer de nouveaux algorithmes et méthodes au projet afin d'arriver à son objectif.

Les principales contributions de cette thèse sont organisées selon trois objectifs:

Détection de défaillance Un algorithme est développé pour identifier un nombre maximal de défaillances de la loi de commande en explorant l'espace des paramètres d'entrée pour un "use case" donné. Pour satisfaire la contrainte industrielle de réduire la puissance de calcul globale nécessaire, c'est-à-dire d'utiliser le simulateur le moins possible, un modèle de forêt aléatoire est utilisé intensivement en tant que modèle "surrogate"

du simulateur dans une boucle d'optimisation avec l'algorithme d'optimisation CMA-ES. De plus, en raison de manque de flexibilité de l'utilisation du simulateur en mode parallèle au moment de ces travaux, un modèle de réseau de neurones est utilisé en tant que modèle de substitution du simulateur actuel, sachant que le terme "simulation" sera néanmoins utilisé.

Détection de frontière Trois algorithmes sont développés pour identifier des scénarios au plus près de la frontière localisée entre zones défaillantes et non défaillantes. En effet, les "use cases" sont généralement définis par des entrées continues, comme les accélérations et les vitesses des véhicules entourant le véhicule autonome. L'espace des paramètres d'entrée peut donc être assimilé à une partition de zones défaillantes et non défaillantes séparées par une frontière à détecter. Chaque algorithme répond à l'objectif d'une manière différente tout en utilisant la même stratégie de réduction de coût de calcul adoptée par l'algorithme de détection de défaillance.

Modèles de frontière Trois approches sont considérées pour identifier analytiquement la frontière aussi précisément que possible dans le but de construire des modèles à la fois performants et explicables par l'intermédiaire d'équations et paramètres: réseau de neurones, programmation mathématique linéaire avec extensions, et programmation génétique appliquée à la régression symbolique. Ils sont tous construits à l'aide de scénarios défaillants et frontaliers tels que ceux identifiés par les algorithmes précédents.

Les algorithmes développés pour les deux premiers objectifs sont testés sur un cas de suivi de véhicule, et leurs résultats sont comparés à la "vérité terrain" calculée sur une grille complète de l'espace des paramètres d'entrée, avec plusieurs métriques utilisées afin de bien évaluer la qualité des résultats obtenus. Les modèles issus du troisième objectif sont testés sur un cas plus sophistiqué avec mise à jour du simulateur, et sont comparés en calculant les erreurs de classification totales sur toute la grille. Tous ces algorithmes et modèles sont injectés au projet ADValue pour aider à la validation du véhicule autonome basée sur la simulation de scénarios.

Acknowledgements

I would like to begin by expressing my deepest gratitude to all the members of the jury, Sébastien Vérel, Jin-Kao Hao, Antoine Grall and Dominique Quadri, for taking part in this thesis committee and making themselves available in such short notice. I sincerely thank my reviewers for their professionalism in providing reviews with valuable comments and suggestions in a record time.

I am deeply indebted to my PhD advisor Marc Schoenauer for his supervision and guidance which greatly helped in the successful completion of this PhD. It was truly an honor to work with you. I am also extremely grateful to my industrial advisors at Renault. Thank you Yves Tourbier for entrusting me with this great opportunity, and Hiba Hage for all the insightful advice during all these years.

I would like to sincerely thank the community of PhD students at Renault, especially Sonia Assou, Matthias Cousté, Benoît Laussat, Lu Zhao and Clara Gandrez for the good times and atmosphere we had at Renault. Thanks to the whole team at Renault, to Martin Charrier who helped with the successful implementation of my work within the ADValue project, and specifically to Joe Antonios for all the great car rides (and venting out during hard times) we shared together across the years. Many thanks should also go to Alex Goupilleau, Ricardo Rodriguez and Assad Balima for the great times (and the pints) during all these afterworks, and to Pierre Caillard and Maroun Khoury for always coming up with interesting lunch discussions.

I also extend my sincere thanks to all the TAU team at Inria and the LRI, mainly Michèle Sebag, Guillaume Charpiat, Corentin Tallec and Diviyan Kalainathan for all the eye-opening discussions and knowledge shared. Special thanks go to Théophile Sanchez who was always a delight to talk to during my time at Inria. Many thanks to Rumana Lakdawala and Félix Louistisserand for their brief but highly appreciated presence at the start of my PhD.

I gratefully acknowledge the assistance of all my friends during all these years. Particularly, I am extremely grateful to Estelle Soueidi for her unparalleled and continuous support that greatly helped me during my PhD journey. I would like to extend my thanks to Cyril Caram for the good times spent during all the meals we shared together. Thanks also go to Maria, Paty, Bahjat, Georges, Yara, Perla, Stéphanie, Hamza, Joëlle, Myriam, Nabiha, Nayla, Jad, Ghina, Joey, Ghady, Sandy, (and many others...) for their support.

Finally, and most importantly, I deeply thank my parents and my sisters Linda and Joanna for their unwavering emotional support and unrelenting help during all these years. I dedicate this thesis to you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Automation in automotive: a brief history

Before the debut of the electric starter in the 1912 Cadillac Touring Edition, a hand crank and a lot of muscle were needed to start driving a vehicle: Electric starter is considered as one of the most significant innovations in the automobile history. The incorporation of this device into vehicles has set the course for automation in the automotive industry in incremental steps. In 1940, Cadillac and Oldsmobile, both General Motors divisions, followed with the first mass-production of a fully automatic transmission intended for passenger use. They dubbed it the "Hydra-Matic Drive", and was considered the greatest advance since the self-starter. Then, in 1958, Chrysler introduced cruise control in motor vehicles. Commonly known as speed control or auto-cruise, it is a system that automatically maintains a selected steady velocity without the use of the accelerator pedal. This system was first available on the Imperial, and was called "Auto-Pilot". Another novel system, the Chrysler "Sure-Brake", was later unveiled in the Imperial in 1971, and introduced a new dimension to brake engineering. It marked the first production of four-wheel slip control system for passenger cars, after thousands of successful stops were tested on various surfaces at maximum deceleration (Douglas and Schafer, 1971).

Nonetheless, the first step toward "autonomous" driving really took place in the 1990s, when innovation really moved up a gear after some trial and error. First, Mitsubishi released the Debonair in 1991, the first worldwide production car to provide a lidar-based distance warning system. Lidar is a distance measuring method that illuminates its target with laser light and measures the reflected light with a sensor. However, this system only warned the driver about vehicles ahead and did not regulate speed, i.e., no influence over throttle, gear shifting or brakes. Four years later, Mitsubishi unveiled yet another breakthrough after equipping its 1995 Diamante with the first laser "Preview Distance Control". This improved system would sense when the closing distance to the vehicle ahead was narrowing, and would automatically regulate speed through throttle control, by easing off the accelerator, and down-shifting to slow down the car. Its key limitation, however, was that it could not operate the

brakes. When the speed difference with the vehicle in front was too great, the only option it had left was to alert the driver with visual and audible warnings. Thus, along with no breaking intervention, other limitations, e.g., poor performance in the rain, velocity operational limit, propelled Mitsubishi to restrict the system solely for the Japanese market, where the road conditions and generally clement weather were more suitable. European markets had to wait until 1999 for a similar system that befits their roads and weather. That year, the Mercedes-Benz S-Class introduced "Distronic", the first radar-assisted Adaptive Cruise Control (ACC). It corrected the biggest limitation of the Japanese system by providing automatic breaking when it detects the car is approaching another vehicle ahead. Its implementation was possible thanks to the development of the computerized Electronic Stability Control (ESC) system by Mercedes that improves vehicle stability by detecting and reducing traction loss. Hence, provisions for automatic braking were already in place. Moreover, Mercedes chose to feature high-quality radar rather than lidar, because it was not affected by rainy, foggy or dusty weather in the way lidar is, and at the same time, was available at a far less expensive cost. Since then, ACC entered multiple iterative improvements to become one of the main foundations of an autonomous vehicle. In the following years, many major automotive manufacturers, also known as Original Equipment Manufacturers (OEMs), introduced their own versions of ACC into their cars, including Jaguar, Nissan, Subaru, BMW, Toyota, Renault, Volkswagen, Audi and Cadillac.

After the successful industrialization of the ACC, other features began to be explored and tested. For instance, the Toyota Prius unveiled its Intelligent Parking Assist System option in 2003. This system combined an on-board computer with a rear-mounted camera and power steering in order to automatically accomplish reverse parallel parking with little input from the driver. Another developed feature is the Automatic Emergency Braking (AEB), which is a system that intervenes independently of the driver, and only in critical situations, to avoid or mitigate a potential accident by applying the brakes. These electronic systems, and many others, are called Advanced Driver-Assistance Systems (ADAS), and are designed to aid the vehicle driver while driving or during parking in order to increase car and road safeties. With the successful production of multiple ADAS in just a few short years, the dream of building a fully autonomous vehicle began gradually to materialize. The race toward achieving this dream, and finishing first, had officially begun. Soon enough, many major OEMs started to develop their own autonomous vehicles without revealing their industrial secrets to the public. Even technology development companies joined the competition later on. For example, Waymo, formerly Google's Self-Driving Car Project, kicked off the development of their own autonomous driving system in 2009 in secret. They tested their software on Toyota Prius vehicles to try to drive fully and autonomously uninterrupted routes. New electric vehicle companies also decided to participate in the race, notably Tesla with its "Autopilot" system. Its first version was introduced in 2015 for Model S cars, and it offered multiple ADAS such as lane control with autonomous steering, self-parking and ACC. The system is also able to receive software updates to improve skills over time, until achieving the delivery of full

self-driving at a future time.

In a nutshell, various companies are currently working on their own versions of autonomous driving software. The reasons behind this competition are numerous, as various inspiring factors encourage the development of a fully autonomous vehicle.

1.2 The benefits of an autonomous vehicle

First, the main reason for developing a fully autonomous vehicle is to increase road safety. Cars equipped with high levels of autonomy have the potential to mitigate risky human driver behaviors, since they are not subject to distraction, fatigue, excessive speeding or impaired driving. Thus, they can help reduce the number of crashes due to human driver errors, which in turn can save money in the process. Fewer crashes can help avoid their costs, e.g., vehicle repair, medical bills and insurance costs.

Next, full automation can offer greater independence for a lot of people. For instance, highly automated vehicles can help people with disabilities to become self-sufficient in their transportation, and can also enhance independence for seniors. They can increase the productivity of drivers to recapture time by doing other activities while the car does all the driving. Plus, better transportation services could see the rise in the form of ride-sharing shuttles that provide more affordable mobility by decreasing personal transportation costs. Self-driving car-sharing systems could transform vehicles from personal propriety to a less costly service called on demand.

Moreover, due to the constant population growth and demographic pressure, increasing traffic density is in the foreseeable future. Autonomous vehicles can help in reducing the number of stop-and-go waves that generate road congestion. Because they also mitigate the number of car crashes on the roads, the resulting traffic jams will decrease as well. This could ultimately lead to a better physical and mental health for passengers of self-driving cars. Another factor is the environmental gains that can result from autonomous driving deployment. Besides playing a role in reducing congestion, car-sharing and automation may spur more demand for electrical vehicles. Hence, greenhouse gas emissions can be greatly diluted from needless idling toward a more environmental-friendly future.

Therefore, autonomous vehicles produce many benefits for the safety, independence, productivity and overall health of passengers on board, as well as the preservation of the environment. Although they can also have certain negative consequences, like causing job losses on a massive scale, namely taxi drivers and lorry drivers, they could also create new positive outcomes like an increased demand for new jobs, such as software developers and high-tech machine experts. However, all these inspiring factors remain theoretical for the moment. When companies decided to put theory into practice and turn fully autonomous driving into reality, they quickly realized that it was impossible to equip vehicles with highly complex systems all at once unless they address the problem in incremental steps by gradually increasing the autonomy of the car.

1.3 Levels of autonomous driving

For the sake of conceiving fully autonomous vehicles, they have been categorized by the Society of Automotive Engineers (SAE, 2018), following levels of driving automation. Each level corresponds to some system complexity that meets the autonomy requirements needed for that level. An Operational Design Domain (ODD) is the set of specific conditions under which a given driving automation system is designed to function, which are different for every level. These conditions include for instance roadway type, traffic conditions and speed range, geographic location, weather and lighting conditions, availability of necessary physical and/or digital supporting infrastructure features, condition of pavement markings and signage...

Six levels of driving automation are defined, from SAE Level 0 (no automation) to SAE Level 5 (full vehicle autonomy) as seen in Figure 1.1.

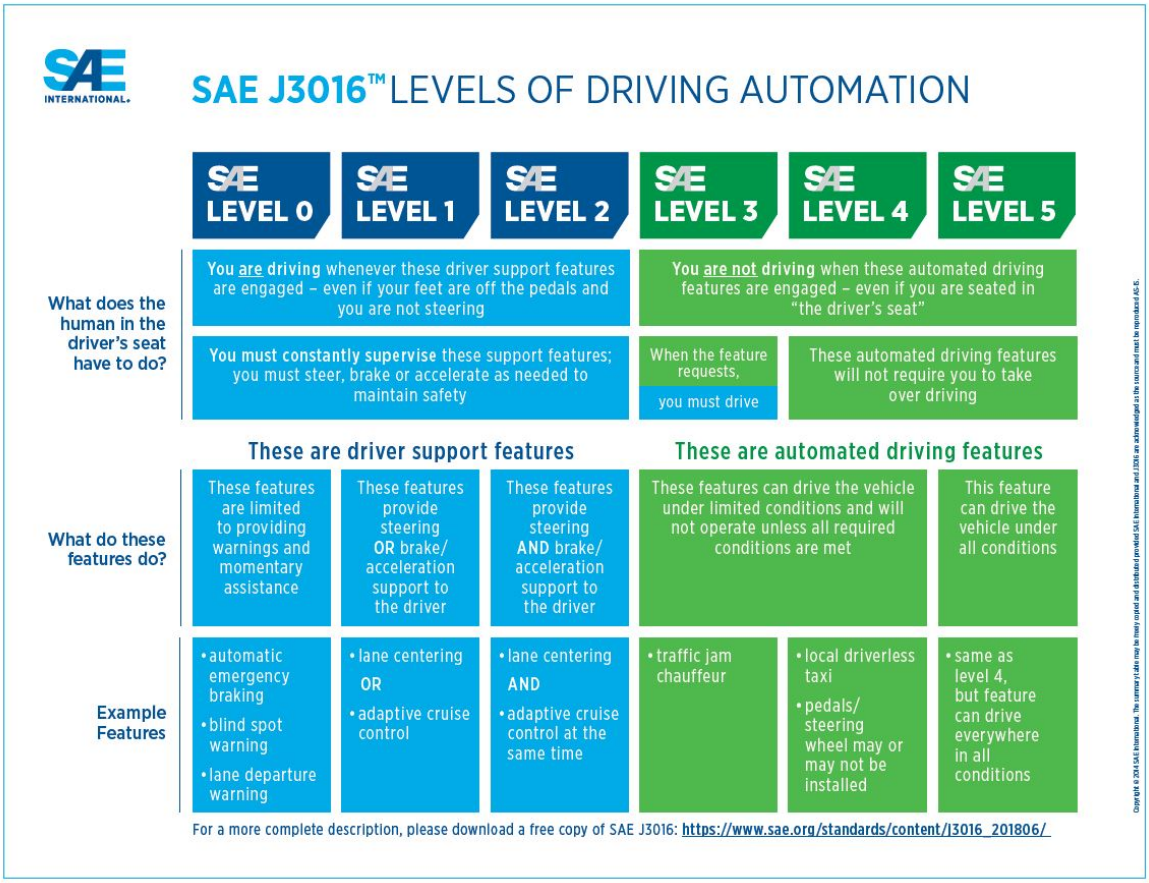

Figure 1.1: SAE J3016 "Levels of driving automation."

Levels 0, 1 and 2 are autonomy systems that can be currently found on vehicles on the roads, whereas levels 3, 4 and 5 are still under research with little to no information about their progress due to the great competition between all companies. We will briefly explain

the driver and vehicle roles for each level.

- **Level 0: No Automation**

  The driver is in charge of all the driving. The vehicle responds only to inputs from the driver. It can, however, provide warnings about the environment. We find here the well-known systems of past car generations e.g., safety belt, warning lights, ESC, and any other system that leaves us completely in control of our vehicle.

- **Level 1: Driver Assistance**

  The driver must still constantly monitor the drive, but gets basic help in some situations. The system can take over either steering (lane centering feature) or acceleration and deceleration, e.g., ACC and Anti-lock Brake System (ABS). The driver must continuously carry out the other.

- **Level 2: Partial Automation**

  The system can now take over both steering and acceleration and deceleration in some driving situations. The ACC and lane centering features can then be used simultaneously. The driver must still constantly monitor the drive even when the vehicle assumes these basic driving tasks.

- **Level 3: Conditional Automation**

  The system still takes over both steering, and acceleration and braking in some driving situations. The difference here is that the driver is not needed to constantly monitor the drive anymore. He can partially be distracted from the road like reading a book or texting, but should be ready to resume control of the vehicle within a given time frame if the system so requests. Therefore, the system should be capable of recognizing its limits and notifying the driver appropriately.

- **Level 4: High Automation**

  The driver can hand over the entire driving task to the system. He can take over the system if it is unable to continue, but he is not required to do so, neither for monitoring, nor as backup. The system can assume all driving tasks under nearly all conditions without any driver attention, and will not operate unless the required conditions are met. Although the features can be the same as those found in Level 3, this level gives greater independence for the driver who can freely pursue other activities than concentrating on the road, like sleeping for example.

- **Level 5: Full Automation**

  The system can take over the entire dynamic driving task in all environments under all possible conditions. The features are the same as those found in Level 4. No human driver is required, i.e., the steering wheel is now optional, and everyone can be a passenger in a Level 5 vehicle.

In that way, the problem is divided into incremental steps of driving automation. When a company successfully completes the technical hurdles of a level, it can continue into the next level and update its systems accordingly. The vehicle is equipped with multiple sensors and algorithms that operate coordinately in order to meet the requirements needed for each level. And because the features requested are more and more numerous and challenging, the sensors and algorithms are also more and more numerous and complex. The system must then be robust enough to handle well all situations. Otherwise, system failures could occur, which in turn could lead to car malfunctions that should not be happening. Besides, the trust of future customers is gained once the new system presented is proved to be fully reliable and surely not prone to cause unexpected crashes. Thus, the testing and validation of the autonomous driving systems is mandatory before industrialization. Testing every component should be assessed intensively in order to mitigate potential failures and avoid unwanted problems on the road.

After all, the autonomous vehicle will be one of the first systems to impact user safety without human supervision. Currently, self-piloted systems in aeronautics continue to be constantly supported for any security action. As for the autonomous railway systems, such as the complete automation systems of the metro lines 1 and 14 in Paris, they operate in a closed and well-controlled environment. The level of responsibility can then be shared between the controlled infrastructure and the autonomous systems, unlike for the autonomous vehicle whose environment is open and shared with the rest of the population. Thus, the validation process of the autonomous vehicle is much more complicated than the testing procedures currently found in aeronautics and railway systems.

The next section details how an autonomous vehicle system is designed and how failures could occur during the process, as well as how companies are currently dealing with all the challenges of building a fully autonomous vehicle.

1.4 The challenges of an autonomous vehicle

1.4.1 Autonomous system description

To become autonomous, the vehicle is equipped with a perception system that maps its environment, a fusion system that synchronizes and combines all sensors for a better object detection, and a decision system that controls the actuators following the fusion objects received to indicate the safest trajectory, as illustrated in Figure 1.2. All systems are briefly explained next.

1. **Perception system**

   This system consists of a set of sensors from various technologies, e.g., radar, lidar, camera, ultrasound, GPS, high definition map or any other existing form of communication technology between vehicles and/or with infrastructure. The sensors are positioned all around the vehicle to give a 360° view of its environment. They can be accompanied

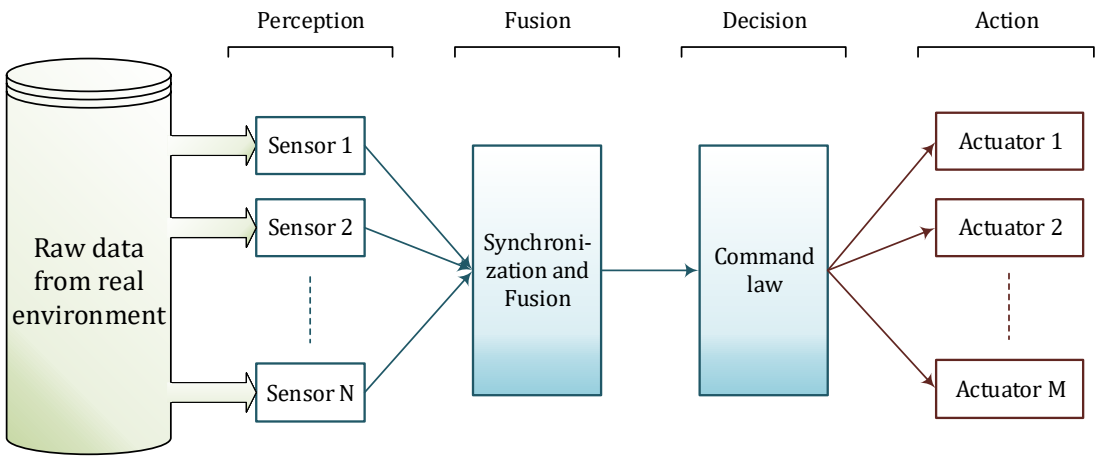

Figure 1.2: Autonomous driving systems flow description.

by an artificial intelligence algorithm which detects elements of infrastructure, side and horizontal road signs, and mobile or motionless road traffic. Each detected object is then tracked, and has a unique identifier per sensor throughout the period during which it is visible to the sensor. If the sensor has seen the object several times, it transmits the information to the fusion system.

However, sensors can give incorrect or imprecise information. They are very sensitive to the real environmental factors, in particular climatic conditions and infrastructure. Therefore, they are usually redundant. The information they transmit is then supplemented or repeated by each other, so as to avoid possible objects omission or false objects detection.

2. **Fusion system**

   This system receives the objects detected and transmitted by all sensors. Because of the redundancy of the sensors, the fusion system can reduce the perception errors. First, it synchronizes the data transmitted by the sensors. Then, it merges the redundant information by taking into account the performance of each sensor. Similar to the sensor algorithms, the fusion algorithm also tracks each identified object. The measurements performed on this object are then mixed with the predictions from previous measurements. A unique and more precise mapping of the vehicle environment is finally obtained, and is transmitted to the decision system.

3. **Decision system**

   This system takes as inputs all final objects, as identified and transmitted by the fusion system. It takes into account any road sign detected, and predicts the trajectories or the

intentions of other traffic vehicles. It computes what the vehicle should do on the road given all the information provided from the vehicle environment following a predefined *command law*. Then, it commands the vehicle of the decisions taken and the desired path in the form of instructions transcribed to the vehicle steering wheel, accelerator, and brakes.

4. **Actuators**

   The actuators are the mechanical parts of the vehicle that perform all requested maneuvers by the decision system.

The autonomous vehicle represents a highly complex system, from the coordination of its various sensors to their fusion, leading to the decision system which ultimately decides the action to be executed by the vehicle. A failure can occur during any stage of this process, which will result in a wrong behavior on the road.

1.4.2 Types of failures

We list a set of different types of errors that can cause the autonomous vehicle to fail, from all different blocks of the vehicle system anatomy to the requirements needed depending on the levels of autonomy.

- **Perception system errors:** As previously mentioned, the perception system is greatly impacted or disturbed by environmental factors, which can cause multiple errors: false detections, non-detections or late detections, measurement errors in objects positions and trajectories, sensor loss resulting in a dangerous stop of the autonomous mode... False detections are commonly named false positives, i.e., an object that does not exist is detected, while non-detections are called false negatives, i.e., an object that should be identified is not detected. All of these errors can cause unexpected damages to the vehicle and its passengers.

- **Fusion system errors:** Since the performances of the sensors vary depending on the environment, a sensor that is generally considered unreliable may be more accurate than others for specific road conditions. The fusion system, whose role is to merge all information transmitted by the perception system to obtain a unique mapping of the vehicle environment, may overlook this sensor and provide an unreliable final mapping to the decision system.

- **Misinterpretation of the decision system:** Beside the bugs that can be found in the development of the decision algorithm, the decision system may misinterpret the inputs provided by the fusion system, such as the intentions of other traffic vehicles, and transmit erroneous commands to the actuators. This typically happens when the decision system faces a new driving situation with a vehicle environment mapping not

encountered before. Its command law does not cover the exact procedures to transmit to the actuators, despite a perfect functioning of the perception and fusion systems.

- **Actuator latency:** The reaction speed of the actuators fluctuates with its environment. For example, the ground can be slippery. The braking speed is then impacted, and the decision system may not be able to anticipate the outcome of the present situation.

- **Conditional and high automations (Levels 3 and 4) emergency situations:** As we have seen in Section 1.3, the system should be capable of recognizing its operating range limits. In the case of level 3 autonomy, the system should notify the driver to take over control of the vehicle, but the driver may not be ready to do so. For level 4 autonomy, the vehicle should handle the situations where the required conditions for its functioning are not met. In both cases, the system must stop the vehicle safely and park it in a safe place. However, this exit may be linked to an end of the authorized use for the autonomous mode, which can result in perturbations in the vehicle behavior. Besides, level 3 is currently described as being the most dangerous autonomy level, because of its conditional status and passing of command between the driver and the vehicle.

These errors are examples that may occur at any time during driving and have then serious consequences on the safety of passengers aboard as well as those involved in road traffic. They are called failures of the autonomous system. It can be impossible to trace which component is responsible for the failure, due to the complexity of the system as a whole. This is why extensive testing and validation are required for each component and at industrialization step.

1.4.3 Validation techniques

As we have noticed so far, the validation of ADAS and autonomous driving systems will occur in highly complex traffic situations. Naturally, software testing is realized on the million lines of codes included in all algorithms of the autonomous driving perception, fusion and decision systems. Possible unwanted software bugs are then eliminated by this source code testing. However, this is not enough to ensure quality testing, as the testing of the autonomous vehicles faces numerous challenges (Koopman and Wagner, 2016; Koopman and Wagner, 2017; Koopman and Wagner, 2018). The vehicle should be assessed in real driving situations to know whether or not it is making the right decisions on the road, and what possible failure implications could follow. However, due to the increase in computing power nowadays, numerical models, simulations and virtual simulators can also be used to test autonomous vehicle functions (Belbachir et al., 2012). These are the main categories of validation used nowadays for the validation of the autonomous vehicle, and are more detailed next. Due to the fast evolution of this field, some publications tried to provide a

comprehensive literature review of the different new safety validation methods within these categories (Huang et al., 2016; Junietz et al., 2018; Riedmaier et al., 2020).

1. **Real test-driving**

   The engineering team at Renault, and every other major car company, rely on real test-driving under various conditions in order to validate the autonomous vehicle. Their main objective is to detect specific system failures during these tests. Then, these failures can be eliminated by updating the system accordingly. This complete test database is conducted at different milestones identified in the design process.

   Two main sub-categories constitute real test-driving methods.

   - **Public road testing:** Autonomous driving cars can be tested in real open environments. For instance, Google mostly carries out real traffic testing for its self-driving cars. Although it is considered useful in going through realistic and probable driving situations easily, it is also very dangerous in case of a failure resulting in a serious accident. Such was the case for the infamous pedestrian fatality involving an Uber test-vehicle in March 2018. The vehicle was operating in autonomous mode in a neighborhood in Arizona with a human backup sitting in the driver seat. Plus, on-road driving of autonomous vehicles requires new regulations and road traffic rules. Some countries, such as Australia, Germany, France, and the United States, began developing such regulations which can allow on-road testing of autonomous vehicles. Nonetheless, a globally harmonized approach has not yet been developed, as there are differences regarding liability and safety provisions (Lee and Hess, 2020).
   - **Closed course testing:** Another way of testing candidate autonomous vehicles in real situations is within closed tracks. There even exists entire testing centers that are dedicated to this method of validation. For example, Mcity is a mock city built for the testing of self-driving cars that is located in the University of Michigan North campus. This technique is safer, obviously, since everything can be controlled on the course. However, it is not scalable, i.e., it can only test driving conditions that humans have thought of and can physically pursue.

   Moreover, real test-driving faces a primordial issue for the long run. In fact, the space of possible driving situations is broken down into use cases. Each use case corresponds to a certain driving situation of the autonomous vehicle, such as an insertion of the autonomous vehicle in a lane, or a cut-in of another vehicle in front of the autonomous vehicle. And for every use case, there are multiple possible scenarios that depend on the traffic situation, road description, environment description, vehicle description... Thus, real test-driving is long and expensive, as it is necessary to reproduce accurately all the imaginable conditions, e.g. weather and traffic, that an autonomous vehicle can encounter. Besides, autonomous vehicles should have a lower accident probability than

human drivers for their launch on the market to be socially accepted (Junietz et al., 2019). Studies have shown that at least 8.8 billion miles (14 billion of kilometers) of test driving are needed to demonstrate with 95% confidence that the autonomous vehicle failure rate is lower than the human driver failure rate (Kalra and Paddock, 2016). Even with a fleet of 100 autonomous vehicles, test-driven 24 hours a day, 365 days a year, this would take about 400 to 500 years.

Hence, it is almost impossible to validate the self-driving cars using real test-driving alone, paving the path for the development of scenario-based validation techniques using models and simulations.

2. **Simulation testing**

   As discussed previously, real test-driving will not be able to cover all imaginable scenarios for the autonomous vehicle in a reasonable amount of time. Simulation testing methods must hence be used for that goal. They are also way cheaper. We differentiate three main methods of validation techniques by simulation.

   - **Resimulation:** Real test-driving data are injected into a numerical model of the command law to try to replicate them in an open loop. It is mostly useful to carry out non regression tests. This will ensure that the algorithm of the decision system still performs well after some upgrade. Nonetheless, closing this loop remains challenging. For instance, we have in our hands a cut-in scenario of a vehicle in front of the autonomous vehicle driving at constant speed. If a new updated algorithm decides the vehicle should accelerate instead, the vehicle could decide not to realize the cut-in eventually. Thus, it would be useful to change the parameters of the real test-driving data in order to predict more realistic scenarios. Nonetheless, the technical hurdles in modifying raw data visuals or sensors objects and trajectories, while trying to stay realistic, remains quite a challenge nowadays. Therefore, closed-loop resimulation remains a research and development subject.
   - **Numerical simulation:** In contrast, this approach is entirely virtual. It generates virtual parameterized data, so it can create new tests which were unavailable in the original test database. The key advantage of this method is that it is a closed loop. The tests are directly launched into a simulation loop, meaning that certain input parameters can easily be changed in order to cover all possible conditions for a certain use case. All past and future events are known at each time step, and can be reproduced and modified. Plus, it is easy to merge multiple simulation models and take advantage of reusing existing results. However, the main drawback of this method is that all results are only valid with respect to the models of the vehicle employed, e.g. all autonomous and dynamic systems, and their degree of realism (Stellet et al., 2015). Further research should determine how to quantify and assess the level of realism of a simulation environment. Until

then, these models are being developed to try to replicate as much as possible the realism of the real systems on the road. Reference by simulation is used, which involves generating synthetically virtual ground truth data that can be compared with real ground truth data to ensure that we have correlation between physical testing and numerical simulation.

- **Virtual simulator:** A person is driving a real vehicle which is connected to its environment through a simulator software using virtual reality glasses or 180° high-resolution screens. This technique is useful in studying the ergonomics and behavior of the driver using autonomous driving mode, while realism of the scenes remains the number one challenge.

Therefore, new and efficient testing methods have been thought through in order to use real test-driving together with simulation to help covering various conditions of the miles needed (Vishnukumar et al., 2017). They are also used to validate each system at a time due to the high complexity of the self-driving vehicle. These available methods are used at each step of the industrial V-model dedicated to the validation of the autonomous driving system (Lakomicki, 2018).

1.4.4 V-model and associated validation strategy

Figure 1.3 shows the V-model that can be applied to the validation of the autonomous vehicle and ADAS features. We will detail each step of the diagram next.

1. **Requirements on the decision system**

   All driving rules, which the autonomous vehicle must respect, are elaborated to satisfy high level functional requirements translated into safety specifications. Then, requirements on decision rules are set, and the behavior of the autonomous vehicle in a given use case is modeled, e.g., its insertion in a lane. This is called Model-In-the-Loop (MIL) testing, which means the model and its environment are simulated with the absence of physical hardware components. The use case is defined by key simulator inputs that characterize the autonomous vehicle environment, and can be modified within a loop to test many possible scenarios generated automatically for a single use case. In that way, we can verify that the actions chosen by the vehicle remain safe according to the driving rules. Hence, these simulations make it possible to test the decision system and numerically validate its requirements while exploring a multitude of scenarios for a defined use case. This is the context of this PhD thesis, and we will get right back to it just after getting through remaining key diagram steps.

2. **Requirements on fusion and perception systems**

   A finer modeling of the data obtained by the fusion system can be added in the MIL to address the requirements on the fusion system. The detection time should be respected,

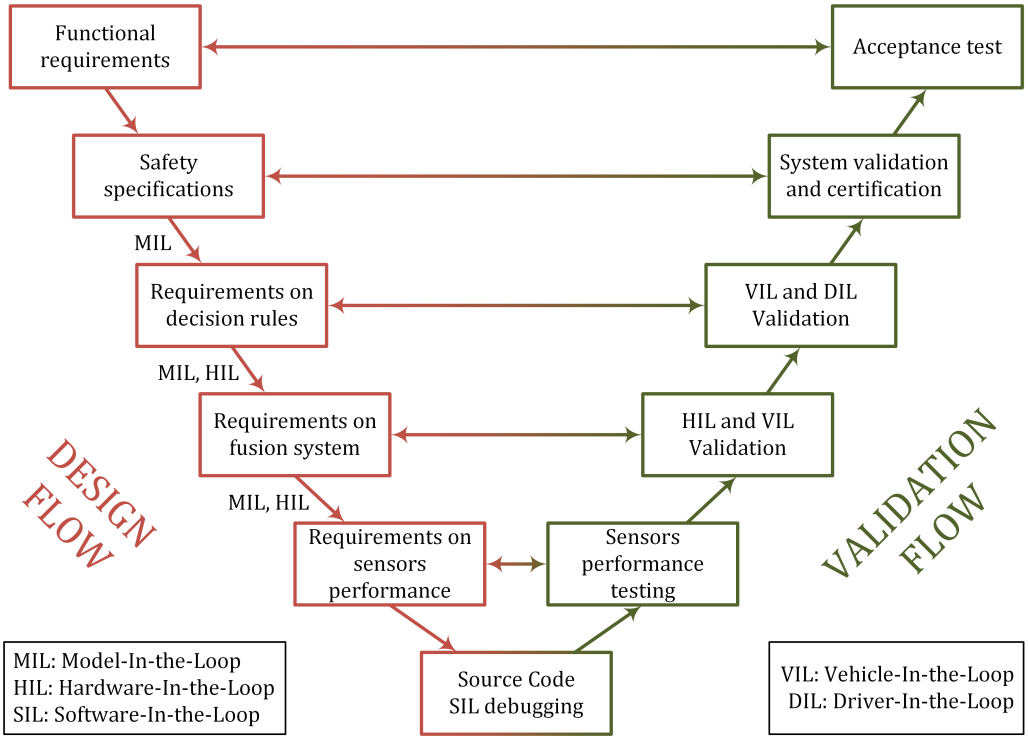

Figure 1.3: V-model of the design and validation flows of the ADAS and autonomous driving perception, fusion and decision systems.

as well as the obstructions of possible objects that could disturb the sensors. It can even integrate perception errors after fusion. For instance, objects measurements could be noisy, and false positives and false negatives could occur. Thus, failure criteria for the fusion system can be deducted, such as a maximum measurement error, or a maximum detection period, not to be exceeded. In order to include latency or other electronic errors, the real electronic components are integrated. This is called Hardware-In-the-Loop (HIL) simulations.

Then, MIL and HIL simulations are completed with sensors objects data by including the sensors algorithms. We are now at the perception system output level. The synchronization and fusion stages are thus evaluated. To verify the vehicle robustness, the sensors errors are also modeled like the fusion outputs. A first validation of the fusion system can be realized so that sensors requirements are defined, such as sensors perception reliability.

3. **Debugging and validation start**

   Software-In-the-Loop (SIL) simulations qualify the reliability of the software implemented in the embedded system by checking for potential bugs and correcting them. At this stage, the design flow concludes and the validation flow begins where the sen-

sors performance is tested by evaluating their electronic hardware reliability and sensors perception accuracy. Although these can be realized by their respective suppliers, the sensors algorithms performance, e.g., objects detection and classification, rely heavily on their integration in the vehicle and on the environment conditions. Tests should thus be conducted on each sensor already integrated in the vehicle to assess their errors and impacts on the proper functioning of the vehicle. This is important because any error of a sensor can have repercussions on the fusion system too. HIL validation is also conducted to better validate the requirements on fusion system.

4. **Decision system validation and final tests**

   The validation of the decision system is not limited to robustness issues linked to a lack of precision in the perception and fusion outputs, but must also integrate the real actions of the vehicle. It must also be able to emit a clear request to the driver to take control of the driving in the case of Level 3 autonomy for example, and guarantee that this intermediate phase takes place normally and safely. Hence, Vehicle-In-the-Loop (VIL) simulations are used to integrate the real behavior of the vehicle actuators by merging on-road testing and simulated elements. The vehicle is tested on a physical track, but the scenario remains numerical through augmented reality while real trajectories are taken into account. Thus, VIL validation is used to further validate the requirements on fusion and decision system simultaneously. Finally, Driver-In-the-Loop (DIL) simulations test the interactions between the driver and the autonomous vehicle. The driver is in a realistic traffic simulator where all his reactions are analyzed during driving, therefore allowing to validate the ergonomics of the human-machine interface while verifying the requirements on decision rules, to verify the security of requests emitted, and to ensure the comfort of the passengers aboard the autonomous vehicle by analyzing their sense of security. Complete system validation and acceptance test are lastly conducted to globally assess the high level safety specifications and functional requirements after completing all past validation steps.

1.5 Industrial context

People at Renault are highly interested in investing in autonomous driving simulation to aid real test-driving in the validation of the autonomous driving mode. A complete validation chain has been defined to address growing technology complexity and use cases diversity. First, use cases are defined through field tests, accident databases, expert knowledge, and global research projects such as Pegasus (Pütz et al., 2017) to form a use case catalog. This catalog defines each use case, as well as the inputs that characterize the parameters influencing the autonomous vehicle environment. Scenarios can then be generated for a single use case by choosing certain values for all inputs, and can be launched into an in-house simulation software called SCANeR Studio (AVSimulation, 2017). It is a software dedicated

to simulation and testing for ADAS and autonomous driving. It provides various tools to run realistic simulations. Numerical vehicle models and sensor models are also built and integrated into the software.

In addition, in order to be able to launch multiple scenarios at the same time, a massive simulation platform is being built. For that aim, thousands of High-Performance Computing (HPC) cores and tens of petabytes of storage capacity will be available by 2025. The SCANeR software as well as all models built are integrated into this platform. We can also integrate the decision system algorithm to the software. Hence, we will have all what it takes to perform a MIL simulation to validate the decision system for a particular use case by 2025 as detailed in previous Section. We set the required software inputs of the use case at the start of the scenario. These inputs are variables that are exterior to the autonomous vehicle, and define its environment. Then, we verify if the autonomous vehicle performs correct and safe actions during the simulation by checking that the output of the simulator satisfies some safety criteria. If not, we have uncovered a failure for this scenario, i.e., this particular combination of inputs for this use case. The decision system should then be corrected in order to take this scenario into account, and avoid obtaining a failure if we run the same simulation of that same scenario with the updated decision system.

Launching a massive simulation plan has become more and more accessible, which is useful to complement real test-driving. As discussed in Section 1.4.3, physical tests can easily validate scenarios that are the most encountered on the road. Simulation should thus be used to cover remaining combinations of input variables that were not tested on the road and detect unlikely edge scenarios, which would otherwise be too costly to validate entirely with real test-driving. However, the dimension of a use case, i.e., the number of input variables that characterize the use case, varies depending on the use case complexity. For some use cases, their dimension can be very high with a wide range of possible values for all inputs. Although possible, simulating all imaginable conditions (up to an acceptable precision for continuous variables) of all use cases can be highly expensive in terms of computing power needed, which is where this thesis comes in.

The goal of this PhD thesis is to facilitate the validation of the decision system by aiding its MIL simulations in exploring the use case input space. In other words, instead of launching a huge number of scenarios and consuming a too large amount of computing power, we want to directly detect combinations of inputs for a particular use case, that will result in a failure of the behavior of the vehicle due to wrong decisions from the decision system during the simulation. These scenarios should include edge cases which represent rare driving circumstances that the actual decision system version has not yet encountered, and can then be eliminated by updating the decision system command law accordingly. Therefore, the complete process of the decision system validation would be to gradually update the command law to decrease the number of failures identified. It is expected that less and less failures would exist as the process goes on – even though the data for only one single step of this overall process (i.e., one single instance of command law) was available at the time of this

work.

This thesis is part of an industrial project at Renault called ADValue (Tourbier, 2017). The project aims at combining multiple algorithms that compete in efficiently exploring the use case input space so as to cover all failure zones in a clever way. In short, our goal is to take a demonstrably smart approach of validating all possible driving conditions while efficiently using time and available resources. In the end, the aim of the thesis is to feed the industrial project with novel algorithms and methods to achieve its objective.

1.6 Objectives of the thesis

All algorithms and models developed during this thesis will be added to the ADValue project in a cooperative way, with the goal of providing a complete tool of numerical validation of the decision system using MIL simulations. They can be divided into three main categories depending on their specific objectives. They will be detailed and addressed in turn in the remaining of this document.

1. **Failure detection**

   This algorithm is developed to explore the input search space for a given use case, and detect as many failures of the autonomous vehicle command law as possible during the simulations. A crucial industrial restriction is added, which is to run as few numerical simulations as possible in order to minimize the computational cost. Thus, the final algorithm should be fast and inexpensive, while being efficient in finding a high number of command law failures caused by scattered input conditions. To achieve this objective, an optimization algorithm and a machine learning model are used through an optimization loop (Nabhan et al., 2019).

2. **Border detection**

   Since the inputs that characterize a use case can be of continuous nature, e.g., velocity and acceleration of the traffic vehicles surrounding the autonomous vehicle, the use case input space can then be seen as a partition of the input space into zones where some combinations of inputs ranges lead to a failure, and others where all scenarios are error-free. A more ambitious objective of the domain experts is then to try to detect the border between such faulty and non-faulty areas, i.e., scenarios that are located near such borders. The rationale behind this is to be able to understand the failures of the command law in depth, and correct it once for whole failure areas, rather than making corrections for one failure after another. Three algorithms are developed for that sake. Each of them addresses the objective in a different way. The models and approach used for the failure detection to reduce computing power are also used here.

3. **Border models**

The previous algorithms detect scenarios that are situated near the border between faulty and non-faulty areas. One step further in the understanding of the failures of the command law consists in identifying the borders between safe and failed zones by simple models, that are both accurate and explainable, to ease further analysis of the failures of the command law. Three approaches are considered to analytically identify the border itself through direct or parameterized equations: neural networks, that are accurate but considered as black-boxes and hence not explainable, and Mixed-Integer Linear Programming (MILP) and Genetic Programming (GP), that both can provide interpretable analytical functions, but might suffer from scaling issues. The three types of models are built using scenarios that have been identified to be on, or very close from, such borders, eventually detected by the algorithms designed for the border detection objective. Their performances are then evaluated by examining their accuracies and total misclassification errors on a full grid covering the input scenario space, and are finally compared.

The remainder of the manuscript is as follows. Chapter 2 surveys the state of the art for the scenario-based validation techniques, including simulation testing, to compare the methodologies used in this thesis with the current approaches found in the literature. Chapter 3 offers a more technical state of the art for optimization and regression techniques, where the methods and models used throughout this thesis are presented in detail. Chapter 4 describes the problem and methods used for the first two objectives, as well as the methodology and results of the algorithm for failure detection related to the first thesis objective. Chapter 5 details the algorithms developed for border detection, the second objective of this thesis. Chapter 6 presents all the methodologies developed to address the third objective, building border models. Finally, Chapter 7 concludes the manuscript and sketches further directions of research.

Chapter 2

Scenario-based Validation

The state of the art review of each technical method used in this thesis is presented in next Chapter. In contrast, this Chapter gives an overview of the current state of the art techniques related to the **scenario-based approach** to assess the safety of autonomous vehicles. This approach focuses on partitioning the driving space into individual traffic situations and testing them by means of virtual simulation. Other validation approaches exist beyond the scope of this thesis (e.g. real-world testing, function-based approach where the system functions are tested based on fixed requirements), and will not be detailed in this chapter. We should also note that, due to the high competitiveness in this field across car companies and tech giants, a complete state of the art review of this subject cannot be achieved. Only the scenario-based methods that are currently accessible are shown, and are compared to our methodologies used in this manuscript.

All publications addressing the scenario-based approach define a scenario to be a sequence of driving events which contain relevant and useful information for the validation process. In fact, when driving on a highway for example, no important action occurs during most of the driving time. Thus, the primary concern of this approach is to detect which scenarios need to be tested, and figure out what is the best method that achieves this goal efficiently. Therefore, two main categories are distinguished in the scenario-based approach.

1. On the one hand, the **coverage-based** methods focus on exploring the driving space, trying to maximize its coverage. They generate new scenario samples, either within input parameter ranges between minimum and maximum values, assuming all scenarios have the same probability of occurrence, or from parameter distributions that add the probability of occurrence of the scenarios in the testing process.

2. On the other hand, the **falsification-based** methods aim at detecting solely edge case scenarios, which are the counterexamples that provoke failures to the autonomous vehicle command law. They can use simulation-based falsification with an optimizer in a feedback loop, or consider other non-simulation selection methods based on accident databases or criticality-based and complexity-based selections.

We will now delve deeper into each sub-category in turn, in order to be able to effectively position the work in this thesis and compare our methodologies with the current methods found in the literature.

2.1 Coverage-based methods

The aim of coverage-based methods is to sample a subset of scenarios that will broaden the assessment on the whole space. They can generate samples using the parameter ranges to cover the entire input space, or parameter distributions in order to focus more on the exposure of the scenarios in the real world.

2.1.1 Parameters ranges

Standard techniques using parameter ranges are ones which consider all possible combinations of the parameters. A coarse discretization is applied on all parameters, transforming all continuous parameters of the use case inputs, e.g., velocity and acceleration values of traffic vehicles, into discrete ones following a step size. Ponn. et al. (2019) use **N-wise sampling** for simple SAE Level 1 functions such as Lane-Keeping Assistants. Nissan and Siemens use **Combinatorial Test Design (CTD)**, originally developed by IBM (Route, 2017), to cover the whole search space. By turning the input parameters continuous space into a finite set of scenarios, any scenario that does not belong to this discrete set will not be addressed. If the step size is too low, one needs to simulate an enormous amount of scenarios, resulting in an increasing computational power, which contradicts the industrial constraints defined by Renault. Hence, the approach is to increase as much as possible the step size to minimize the resulting number of simulations. This technique has the potential to quickly determine different failed scenarios over the input space. However, if the step size is too high, a lot of critical scenarios could be overlooked and failed to be detected. Tuning the step size for each use case remains the key disadvantage of these methods, which caused other techniques to spawn.

Rocklage et al. (2017) presented an **automated framework for regression testing**. It automatically generates and combines variations of multiple parameters, e.g., road, static objects, dynamic objects, environmental conditions, by ensuring they are physically reasonable using a combination of a backtracking algorithm and a trajectory planner. They showed that the algorithm is operational, but is not fully efficient yet. It needs further optimization, and it can only generate scenarios according to predefined paths. Majzik et al. (2019) use **Signal Temporal Logic (STL)** to assess the autonomous vehicle at the system level. They start with an existing test set which is in compliance with the safety standards regulations, and aim to increase coverage by creating new test cases using graph generation techniques. **Satisfiability Modulo Theories (SMTs)** are used to sample a full road network from multiple criteria (Kim et al., 2016, 2019). The SMT-solver can generate road segments based on a definition of the curve coverage criteria and some constraints. Khastgir et al. (2017)

simply use **randomization techniques** to create new scenarios by applying them to the brake and accelerator pedal values of the autonomous vehicle.

Huang et al. (2018) create new scenarios for a SAE Level 2 autonomous vehicle by adding surrounding vehicles, modifying their continuous parameters, e.g., starting position, velocity, lateral and longitudinal behaviors, and combining them all into a large data set. Then, they apply a **Scenario Importance** strategy in order to reduce this high number of scenarios by eliminating ones below a threshold value. This method aims at studying the impact of the surrounding vehicles on the behavior of the autonomous vehicle, as their influence is proportional to the Scenario Importance factor, and is applied on a curve driving situation. Similar approaches consider focusing on surrounding traffic as well. Xie et al. (2018) implement this method in three different driving situations: tracking, curve and lane-change, while Zhou and del Re (2017) apply it for an Adaptive Cruise Control (ACC) assessment by taking into account the number of participants and their abstract behavior in order to generate a structured test catalogue.

Additionally, Beglerovic et al. (2017) use a **Design of Experiments (DoE)** as an initial test design, which samples scenarios following a certain rule everywhere in the search space. Then, it is used for optimization purposes depending on the desired goal as we will see in Section 2.2. Our methodology chosen in this thesis stems from a similar approach by also considering a DoE from which a small dataset is initially drawn. Subsection 4.4.2 will show the use of low discrepancy sequences which can generate a DoE that is able to fill the space of possibilities more evenly than other similar techniques (Santiago et al., 2012).

Finally, **Rapidly exploring Random Trees (RRTs)** are used by Althoff and Dolan (2012) to generate trajectories which will help them assess the results of reachability analysis, another validation technique, and are shown to achieve good coverage of the search space. Tuncali and Fainekos (2019) also use RRTs to define a custom function based on various parameters, e.g., collision surface and velocity, in order to determine boundary scenarios which witness the transition from safe driving to collision occurrences. They create trajectories that lead to collisions and are able to avoid local minima while reaching the global minimum. Chapter 5 also tackles the detection of border scenarios between faulty and non-faulty areas, which represents the second objective of this thesis: We will consider different approaches and define three algorithms relying on an optimization loop. These algorithms attempt to detect border scenarios everywhere in the search space, without the need of generating trajectories. They are also developed to detect the border of any evaluation criterion wanted, whereas Tuncali and Fainekos (2019) can only apply their method on collision-related criteria.

2.1.2 Parameters distributions

In this thesis, we mainly focused on sampling with parameters ranges which define the driving situation in the simulation software. We will then proceed by briefly describing alternative methods found in the literature that use parameters distributions instead. These approaches rely mainly on **Accelerated Monte Carlo** methods. In fact, Monte Carlo

techniques can be used to generate new samples by estimating the expected probability of an event happening, which is the failure probability in our case, but can take a long time to execute if implemented in its basic random way, which makes it inefficient. Thus, accelerated Monte Carlo techniques are introduced in the next papers presented to sample scenarios from parameters distributions, and are compared to the standard Monte Carlo method sampling.

A first accelerated approach uses **Extreme Value Theory** (Åsljung et al., 2016, 2017). The authors evaluate the system safety level based on real data and a criterion metric by relying on near misses situations, i.e., narrowly avoided collisions. It is noted that the chosen criterion metric can greatly impact the level prediction before considering the Brake Threat Number as a promising criterion metric. They further elaborate that this method is more efficient than a statistical approach which needs 45 times more measurement data.

Importance Sampling Theory is also used to quickly predict the failure probability while maintaining statistical correctness and accuracy (Zhao, 2016; Zhao et al., 2016, 2017; O'Kelly et al., 2018). In one publication, the authors chose to apply the technique on a lane change use case, and select three variables which they consider are the only ones necessary to mostly characterize a possible vehicle collision. The parameters distributions are derived from data collected from vehicle driven under naturalistic conditions. While they were able to reach the same accuracy faster than the naturalistic tests in simulation, the main downside of this technique is that it is unsuitable for big use cases where more than a hundred of parameters are required to describe the driving situation. A high amount of simulations is then needed to cover the distributions identified for all parameters, which in turn increases the computational power required for these simulations. Nonetheless, one advantage of this method is observed when Importance Sampling is combined with another validation analysis in order to sample scenarios that are physically feasible and realistic (Wang et al., 2020). Furthermore, this method spawned other works and publications that are presented next.

Huang et al. (2017b) model the vehicles behavior using **Piecewise Mixture Distribution** and apply it on a cut-in driving situation (Huang et al., 2018). The results show that the method is 7000 times faster than the standard Monte Carlo method. Plus, another technique is based on **Kriging Models** and tackles the problem by proposing a sequential learning approach (Huang et al., 2017; Huang et al., 2017a). It searches for the next best scenario by relying on a heuristic simulation-based gradient descent procedure. It is able to decrease experimental runs and save on-track experimentation, while being more efficient than random scenario sampling despite a lack of quantitative statement. Similar work from the same research group is available too (Arief et al., 2018; Zhang et al., 2018b; Huang et al., 2019). ANSYS is also recently employing Kriging models to focus on corner cases while performing closed-loop testing of the autonomous vehicle. However, the main drawback of Kriging models is their incapacity in handling use cases of high dimension, or ones with qualitative inputs, e.g. vehicle type and road type. Nonetheless, some extensions of Kriging models exist in the literature. They have been tried out in the aim of bringing a broader scope of applicability for Kriging models (Echard et al., 2011; Chen et al., 2013).

A last accelerated technique used is the **Markov Chain Monte Carlo** sampling, which does not require a quantitative evaluation of the overall risk level (Akagi et al., 2019). The parameters distributions are provided by real data similar to the previous accelerated methods presented, and a traffic risk index evaluates the scenarios with their corresponding parameters. However, unlike previous methods, the tested scenarios were not reversely calculated to assess the overall safety evaluation. Furthermore, this method is also used to develop a road generator (Olivares et al., 2016), and highlight the influence of the discretization of the traffic participants states on the resulting criterion metric errors measurements (Åsljung et al., 2019).

In conclusion, although parameter distributions can provide a deeper significance of the scenarios in the real world, the accelerated Monte Carlo methods are limited by the required computation power for complex driving situations. Plus, they are usually based on real world data which are not easily accessible due to the high competitiveness of the field. Therefore, Renault opted for the characterization of simulated driving situations using parameter ranges for now, which in turn is naturally used in this thesis work.

2.2 Falsification-based methods

As mentioned previously, the falsification-based techniques aim at detecting edge cases, which are failed scenarios that the autonomous vehicle system has yet to encounter. The main technique is to rely on simulation while building an optimization loop, whereas other techniques are also derived via non-simulation approaches. Since our work in this thesis belongs to the simulation-based techniques, we shall begin by briefly reviewing the non-simulation approaches before delving deeper into the simulation-based methods to set baselines for our work, and position our methodologies more effectively.

2.2.1 Non-simulation detection

Accident databases

Accident data are primarily used to test driver-assistance systems. The scenarios, where the human driver performance leads to an accident, are retrieved in order to update the automated vehicle systems and increase their safety.

For instance, the GIDAS accident database is used to evaluate the requirements needed by the driver-assistance systems to mitigate potential accidents in urban zones (Stark et al., 2019; Stark et al., 2019), and in limited access highways (Feig et al., 2019). Others rely on accident databases to form the basic starting point for a scenario-based approach validation using simulations (Fahrenkrog et al., 2019), or Big Data techniques that rely on the words most frequently mentioned in the crash descriptions (So et al., 2019). They derive new critical scenarios from accident data to generalize the potential of the system to avoid accidents, and validate its effectiveness.

Nonetheless, data derived from accidents cannot be used extensively for the validation of an autonomous vehicle of SAE Level 3 and higher. The system is able to rely on them to be updated by mitigating accidents that already happened, but should turn to different techniques to deal with accidents that have not yet taken place, to try to avoid that they occur in the future.

Criticality and Complexity

Two other non-simulation approaches are defined in order to generate new edge cases scenarios: **criticality-based** and **complexity-based** selections.

In the first approach, the **criticality-based** selection, the idea is to increase the criticality of safe scenarios. Pierson et al. (2019) focus on calculating the risk that the autonomous vehicle encounters by looking at the position and velocity of other traffic vehicles behaviors. They select highly critical scenarios (i.e., for which they calculated high values of risk), and evaluate them using the HighD (Krajewski et al., 2018) and NGSIM (FWHWA and US Department of Transportation, 2017) real data sets. In a similar way, another method calculates the risk by relying on the safe area that the autonomous vehicle can use based on the behaviors of the surrounding vehicles. They resort to evolutionary optimization to minimize this safe area, which also maximizes the scenarios criticality (Althoff and Lutz, 2018; Klischat and Althoff, 2019). However, the generated scenarios have not been reversely calculated to check if they have indeed created scenarios where the autonomous vehicle system would fail.

In the second approach, the **complexity-based** selection, the idea is to consider the parameter ranges of a driving situation and detect critical scenarios by increasing the complexity of the scenario. For instance, the Analytic Hierarchy Process is used to assign weights to each scenario parameter, which in turn defines a complexity index later combined with combinatorial testing (Xia et al., 2017, 2018; Gao et al., 2019). The procedure is evaluated on a warning system designed for lane-departure, and the results show a correlation between scenario complexity and number of system failures. Wang et al. (2018) define the complexity by separating it into two subcategories; the first one defines the static environment complexity, whereas the second one describes the dynamic environment complexity, i.e., taking into account the behavior of the other vehicles. Nonetheless, this metric was not validated. Alternatively, Zhang et al. (2018a) describe the complexity related to the cognitive capabilities and tasks of the autonomous vehicle by taking into consideration challenging weather conditions, e.g. fog, and road type and structure. Their results indicate that scenarios complexity and performance are negatively correlated. Qi et al. (2019) define the complexity through a Scenario Character Parameter (SCP) applied to trajectories resulting in system errors. These trajectories are then grouped together into similar clusters in order to extract the most important and challenging cases. Other metrics which impact the scenarios complexity can be found in the literature (Guo et al., 2019; Koopman and Fratrik, 2019).

2.2.2 Simulation-based detection

The simulation-based approaches differ from the previous methods presented as they need to have access to a simulator in order to propose new edge cases scenarios. For that matter, they need an optimizer which selects the next scenarios, then forwards them to the simulator to be executed. After evaluating the new scenarios, the optimizer takes the results into consideration in order to determine the next scenarios for the iteration that follows. Depending on the cost function considered, the optimizer detects more critical scenarios throughout the iterations for the driving situation considered with the autonomous vehicle. Multiple methods are shown next, which mainly differ in the choice of optimizer and methodology used.

Koren et al. (2018) are part of a research group which uses **Reinforcement Learning** and calls the procedure Adaptive Stress Testing. They build upon the results of a publication in the avionic domain (Lee et al., 2015), which inspired a learner to compare two simulators by maximizing the deviation between them (Lee et al., 2018). They use Deep Reinforcement Learning along with Monte Carlo Tree Search to generate trajectories of pedestrians and sensor noise in a driving situation consisting of an autonomous vehicle getting close to pedestrians on a crosswalk. They are able to focus on both actions and sensor failures throughout the time steps. From the same research group, Corso et al. (2019) deepen the technique by introducing a reward-augmentation method. This procedure is promising according to the publications. However, it has been tested so far on simple driving situations and has yet to be assessed for more complicated and time-consuming cases.

Another technique used for simulation detection is the closest to our methodology in tackling the first objective of this thesis, that is, developing an algorithm for failures exploration. Beglerovic et al. (2017) present a loop involving **surrogate modeling** using Kriging and **stochastic optimization** consisting of Differential Evolution (DE) genetic optimization (Storn and Price, 1997a) and Particle Swarm Optimization (PSO) (Poli et al., 2007) algorithms. Just as we will describe the problem statement in Subsection 4.1, a surrogate model is introduced to handle the launch of simulations necessary for optimization iterations instead of relying on the costly simulation engine. This model is then updated with the global minimums found to improve its accuracy. A so-called zooming-in algorithm manages the whole interactions between the surrogate model and the simulation software. This method is based on another procedure which uses neural networks instead of Kriging as surrogate models (Ben Abdessalem et al., 2016), and was able to reduce the number of simulations needed to discover faulty scenarios. However, Beglerovic et al. (2017) opted to choose Kriging instead of neural networks as surrogate models because, on one hand, they lacked expert knowledge in deep learning to build a satisfactory model, and on another hand, wanted a surrogate model without lengthy training needed nor heavy data preparation beforehand.

Multiple other publications present similar work while developing alternative approaches. For instance, **Range Adversarial Planning Tool (RATP)** is a methodology used to generate new scenarios by the means of adaptive search-algorithms based on the previous results obtained (Mullins et al., 2017; Mullins et al., 2018; Mullins, 2018). Other global

optimum search techniques used involve **Bayesian optimization** (Gangopadhyay et al., 2019) or **simulated annealing** to assess perception algorithms (Abbas et al., 2017).

Tuncali et al. (2016) developed **S-TaLiRo**, which is their own automatic test generation engine based on formal system requirements. They first used simulated annealing as means of optimization before switching to gradient descent with greater performance (Tuncali et al., 2017). They define the cost function as the gap of a criticality metric quantified to falsify the formal system requirements. They applied their technique on a velocity control system while attempting to avoid a collision. Finally, Koschi et al. (2019) separate detecting failures in the ACC system into performing **forward and backward searches**. The latter one involves starting from a simulated accident and optimizing it backwards in time, which can detect a fault more efficiently than the forward-search method.

2.3 Problem setting

In this thesis, we will present in Section 4.5 the methodology used for the first objective of failures exploration, and in Sections 5.1 and 5.2 the methodologies used for the second objective of border detection. In both objectives, a random forest model is used as substitution model of the simulator in our optimization loop. Furthermore, the main difference between previous studies and the work presented for the first objective, is that the detection of edge cases is optimized to detect higher probability of failures, yet little research was found that attempted to tackle coverage and falsification by the same algorithm. While some studies tried to combine combinatorial testing for coverage and simulated annealing for failure detection (Tuncali et al., 2018; Tuncali et al., 2020; Tuncali, 2019), their ultimate goal was different as they aimed at improving the optimizer convergence by enhancing its initialization through the selected scenarios. In fact, Felbinger et al. (2019) perform a comparison between combinatorial testing and failure detection, and show that both methods are able to detect critical scenarios without efficiency-assessing.

In contrast, the first algorithm developed in this thesis aims at detecting failed scenarios while simultaneously exploring the whole space of all driving situations. Thus, coverage and falsification are tackled simultaneously. This algorithm corresponds to the **first** objective of this thesis, and is applied on a tracking vehicle use case, where its efficiency is evaluated by how much it succeeded in covering edge cases for the whole space. Then, the three algorithms developed for border detection address the **second** objective of this thesis, and show different approaches than the RTTs used by Tuncali and Fainekos (2019) which only tackle collision events by generating trajectories. They are developed to detect border scenarios for any evaluation criterion defined and without the need of creating trajectories. Additionally, the computational budget of these four algorithms is limited in terms of number of simulations. Such industrial constraint is imposed to limit the overall computational cost, forcing the algorithms to reach their objectives in a cost-effective way.

Lastly, no noticeable publication was found that worked closely on the **third** thesis ob-

jective, i.e., building representative models of the input space border based on scenarios close to the border. The main goal associated to this objective is to better understand the failures of the command law than with only single failure points, so as to hopefully be able to correct it more globally. Explainability of Machine Learning models is a hot topic today (DARPA launched the XAI program 4 years ago), and two approaches exist. The first one, whose flagship is LIME (Local Interpretable Model-agnostic Explanation) (Ribeiro et al., 2016) builds, at its name suggests, local model of any given learned model that can explain one decision, or all decisions in a small part of the space. But this is not what is wanted here, and global explanations are totally domain- and model-dependent. This is why specific work is needed here too. The border models that will be built in this final part of the thesis should be as explainable as possible, while representing as accurately as possible the border as per the ADValue project specifications.

Now that we have positioned our work within the scenario-based validation methods found in the literature, the next Chapter is devoted to give a comprehensive state of the art review of all technical methods used throughout this thesis.

Chapter 3

Optimization and Regression

This Chapter is dedicated to brief surveys of the algorithms that will be intensively used throughout this thesis: derivative-free optimization algorithms, that aim at optimizing black-box real-valued functions defined on real-valued or mixed search spaces, and regression algorithms, whose goal is to learn in a supervised way models of real-valued functions from input/output examples.

3.1 Optimization Algorithms

This section briefly introduces some optimization algorithms that will be used in this thesis, and gives some motivation for their choices. By no means this presentation pretends to be exhaustive, and a few other algorithms could certainly have been chosen with similar properties without modifying the main conclusions of this thesis.

The CMA-ES algorithm is used twice in this manuscript: the **Find One Failure** algorithm in Section 4.5.4 uses CMA-ES as optimization engine to detect a failed scenario the farthest possible from an archive of failed scenarios, and the **Find One Border Point** algorithm in Section 5.2.1 uses CMA-ES to locally detect a scenario on the border between failed and safe areas starting from an initial point. Then, Mixed-Integer Linear Programming is considered as a possible approach for border models in Section 6.3. Several models have been tested leading to a final industrialized version.

3.1.1 CMA-ES, a Derivative-Free Stochastic Optimization Algorithm

Derivative-free algorithms will be needed throughout this work. Several metaheuristics, i.e., stochastic optimization algorithms, belong to this class of algorithms, though not all metaheuristics deserve attention (Sörensen, 2013). Among them, Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995), Differential Evolution (DE) (Storn and Price, 1997b) and Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen and Ostermeier, 2001) have been demonstrated to be efficient. However, though slightly more com-

plex, CMA-ES has also won many challenges against a load of competitors, beyond just PSO and DE: see for instance (Auger et al., 2009), in which CMA-ES is experimentally compared to PSO and DE but also to BFGS (with numerical differentiation) and NEWUOA (Powell, 2006), a recent deterministic algorithm for derivative-free optimization. Another family of algorithms derived from Bayesian optimization could also have been used here, in particular the EGO algorithm (Jones et al., 2006). However, Bayesian Optimization scales very poorly with the dimension of the search space, whereas CMA-ES does scale up to around a few hundred variables (and even more with specific large-scale variants). Thus, even if this work does not address problems with large dimension, it should be used in the future for real-world problems where Bayesian Optimization might suffer. Hence the choice for this thesis of CMA-ES, that we are going to briefly introduce.

The algorithm CMA-ES (Hansen and Ostermeier, 2001) is a stochastic optimization algorithm for continuous optimization problems. Originated from the Evolutionary Algorithm domain, it is a derivative-free optimization algorithm, i.e., it doesn't need any assumption on the objective function to optimize.

CMA-ES evolves a multi-variate Gaussian distribution $\sigma\mathcal{N}(m, \boldsymbol{C})$ defined by its mean m, a strictly positive multiplicative factor σ, called the *step-size*, and its covariance matrix $\boldsymbol{C}$ (by abuse of notation, as the actual covariance matrix is in fact $\sigma^2\boldsymbol{C}$). At each iteration, a set of λ points of $\mathbb{R}^n$ is sampled according to this multi-variate Gaussian distribution. Evaluation of the objective function, aka *fitness*, of the different points is performed and the parameters of the Gaussian distribution are updated: the step-size is increased (resp. decreased) if successive steps of the algorithm tend to go in the same direction (resp. have a sum close to 0); the covariance matrix is modified so as to favor other step in the direction of past successful steps.

More precisely, from this Gaussian distribution λ offspring are sampled independently:

$$x_k = m + y_k, \text{ with } y_k = \sigma\mathcal{N}_k\left(0, \boldsymbol{C}\right) \text{ for } k = 1, \ldots, \lambda$$

The fitness of all offspring is computed, and the linear combination of the μ best offspring becomes the new mean:

$$m \leftarrow \sum_{i=1}^{\mu} w_i x_{i:\lambda} = m + \sigma y_w \text{ with } y_w = \sum_{i=1}^{\mu} w_i y_{i:\lambda} \tag{3.1}$$

where the positive weights $w_i \in \mathbb{R}$ sum to one. The index $i{:}\lambda$ denotes the i-th best offspring according to the fitness values.

The strategy parameters of the algorithm, the covariance matrix $\boldsymbol{C}$ and the step-size σ are updated so as to increase the probability to reproduce good steps. The so-called rank-one update for $\boldsymbol{C}$ takes place as follows. First, an evolution path is computed:

$$\vec{p}_c \leftarrow (1 - c_c)\vec{p}_c + \frac{\sqrt{c_c(2 - c_c)\mu_{\text{eff}}}}{\sigma} y_w$$

where $c_c \in]0,1]$ is the cumulation coefficient and μ_{eff} is strictly positive. This evolution path can be seen as the descent direction for the algorithm. Second, the covariance matrix $\boldsymbol{C}^{(g)}$ is "elongated" in the direction of the evolution path, i.e., the rank-one matrix $\vec{p}_c\,(\vec{p}_c)^T$ is added to $\boldsymbol{C}^{(g)}$:

$$\boldsymbol{C} \leftarrow (1 - c_{\text{cov}})\boldsymbol{C} + c_{\text{cov}}\vec{p}_c\,(\vec{p}_c)^T$$

The complete update rule for the covariance matrix is a combination of the rank-one update previously described and the rank-mu update presented in Hansen et al. (2003). It extends the previous rank-one rule the following way:

$$\boldsymbol{C} \leftarrow (1 - c_{\text{cov}})\boldsymbol{C} + c_{\text{cov}}\vec{p}_c\,(\vec{p}_c)^T + c_\mu \sum_{i=1}^{\mu} w_i y_{i:\lambda} y_{i:\lambda}^T$$

The update rule for the step-size $\sigma^{(g)}$ is called the path length control. First, the following vector is computed:

$$\vec{p}_\sigma \leftarrow (1 - c_\sigma)\vec{p}_\sigma + \sqrt{1 - (1 - c_\sigma)^2 \mu_w}\,\sigma \times \boldsymbol{C}^{-\frac{1}{2}} y_w$$

where $c_\sigma \in]0,1]$. The length of this vector is compared to the length that would have had this vector under random selection, i.e., in a scenario where no information is gained from the fitness function, and one is therefore willing to keep the step-size constant. Under random selection, the vector $\vec{p}_\sigma$ is distributed as $\mathcal{N}(0, Id)$. Therefore, the step-size is increased if the length of $\vec{p}_\sigma$ is larger than $\mathbb{E}(\| \mathcal{N}(0, Id) \|)$ and decreased if it is shorter. In practice, the update rule reads:

$$\sigma \leftarrow \sigma \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\| \vec{p}_\sigma \|}{\mathbb{E}(\| \mathcal{N}(0, Id) \|)} - 1\right)\right) \tag{3.2}$$

where $d_\sigma > 0$ is a damping factor.

The default parameters for the strategy are given in Hansen and Kern (2004) Eqs. 6-8. Only the initial values for the mean and the step-size have to be set depending on the problem.

In the default setting, the population size λ is equal to $\lfloor 4 + 3\log(n) \rfloor$, where n is the dimension of the search space. Increasing the population size increases the probability to converge towards the global optimum when minimizing multimodal fitness functions (Hansen and Kern, 2004). This fact was exploited in Auger and Hansen (2005) with a CMA-ES restart where population size is doubled after each restart is proposed. Different stopping criteria are used to determine when to restart the algorithm, the main one, and only one used in this thesis, is the tolerance on fitness: the algorithm stops if the range of the best objective function values of the recent generation is below a given tolerance value.

CMA-ES not only does not need any gradient of the fitness function, but also it uses a ranked-based selection, and is therefore invariant when optimizing a fitness function f :

$\mathbb{R}^n \to \mathbb{R}$ or $h \circ f$, where $h : \mathbb{R} \to \mathbb{R}$ is any ranked preserving transformation. This property implies in particular that convexity does not play any role for the convergence of CMA-ES.

Finally, it is worth noting that CMA-ES is a completely Open Source software, and the source files in several programming languages, including Python, Matlab, and C, are available on the main authors's web page at http://www.cmap.polytechnique.fr/~nikolaus.hansen/, together with a detailed tutorial and links to all related publications, including hundreds of applications of the algorithm to different domains.

3.1.2 Mixed-Integer Linear Programming

In a nutshell, linear programming is a method used to solve a linear program, i.e., to minimize a linear objective function subject to linear equality and inequality constraints as well as positiveness constraints on the variables. Linear programming dates back in the nineteenth century, but modern algorithms that are still used today were introduced in the second half of the twentieth century (see (Orden, 1993; Gass, 2002) for more historical details).

Linear Programs The canonical form of a linear program is the following:

$$\begin{cases} \min_x c^T x \\ Ax \leq b \\ x \geq 0 \end{cases}$$

where $c \in \mathbb{R}^n, b \in \mathbb{R}^n$ are known n-dimensional vectors, A is a $p \times n$ matrix, and the notation $z \leq 0$ for any given vector $z \in \mathbb{R}^n$ means that all components of z are negative. Equality constraints are transformed in 2 inequalities in A. Vector $x \in \mathbb{R}^n$ are the decision variables.

The feasible region (set of points that satisfy the constraints) is the polytope defined by the inequalities $Ax - b \leq 0$, intersected with the positive quadrant. It is convex, and hence, if it is not empty and bounded, there exist solutions to the LP. Moreover, because the objective function is linear, these solutions lie on the boundary of the feasible region.

Several methods have been proposed to solve the general linear program. The earliest generic and efficient algorithm is the so-called *simplex algorithm* that explores the vertices of the polytope defined by the inequalities $Ax - b \leq 0$, iteratively moving along its edges. The main disadvantage of the simplex algorithm is that its worse case complexity is exponential - tough in many practical instances, it is polynomial. The other popular methods today to solve a linear program are the different *interior-points* algorithms: First proposed in 1984 (Karmarkar, 1984), these algorithms, as their name indicates, also explore the interior of the polytope. They are probably polynomial, and perform well on most practical instances. Currently, the majority of software solutions are built around the simplex algorithm and interior-point algorithms.

Integer Linear Programs and Mixed Integer Linear Programs An integer linear programming (ILP) problem refers to a linear program where all decision variables are constrained to be discrete. If all variables are discrete, the model is called a pure integer program. Otherwise, the model becomes a Mixed-Integer Linear Program (MILP). ILPs are known to be NP-hard, hence no polynomial time algorithm exists to solve them. As a subset of ILPs, MILPs share the same property.

In particular, the relaxation of a MILP, in which the integer variables are allowed to take values in $\mathbb{R}^n$ is not a solution per se: it can be solved in polynomial time (as all LPs on $\mathbb{R}^n$), but there is no reason why the solution of the ILP could be the rounded solution of the relaxed LP. However, using the relaxed LP together with a Branch-and-Bound strategy on the integer variables is the basis for the most popular approaches for solving MILPs, used in most available software today: the algorithms iteratively solve a relaxed problem, and add two branches to the search tree by adding constraints on one variable on both side of its (continuous) value at the solution of the relaxed problem. Additional pruning of the search tree is provided by the solution already explored.

Two softwares will be used in this work to address MILPs in order to derive border models (Section 6.3): the Open Source package `lp_solve` (Berkelaar et al., 2004) and the commercial software Gurobi (Gurobi Optimization, 2020).

3.2 Regression Algorithms

Given a set of iid labeled tabular data, i.e., a set of data points generated by the same process and described by a vector of features (discrete or continuous values), with a label attached to each example, supervised learning aims at building a predictive model of the process at hand, that can predict the label of unseen (hence unlabeled) examples generated by the same process. Regression is one name of supervised learning when the labels are real-valued (and classification is used for discrete labels, aka classes). The learning phase (learning of the model from the set of training examples) usually aims at minimizing the error made by the model by comparing, for each training example, its output with the known labels, aka the *loss function*. Most current losses amount to least squares error, though different criteria can be targeted, leading to different loss functions.

But the ultimate goal of supervised learning is to be able to generalize the learned model to unseen examples, i.e., to minimize the error on the whole example space: the standard procedure to estimate this unreachable ideal error is to learn the model on some subset of the initial sample set, aka the *training set*, and to keep part of the initial sample set hidden from the learning phase: the error of the learned model on this *test set*, for which the true labels are known, is a measure of the generalization error of the learned model.

Furthermore, all learning algorithms come with their own hyperparameters that need to be tuned anew for each new problem. And there is generally no theoretical result that could guide these choices. The standard experimental way is to split the original dataset

into three subsets: the training set and the test set, already mentioned; and the *validation set*, that is used to tune the hyperparameters, and should be distinct from the training and the test sets. Learning is performed on the training set using different settings of the hyperparameters, the set of hyperparameters that performs best on the validation set is chosen, and the generalisation performance is then evaluated on the test set with these 'optimal' hyperparameters. Of course, such procedure assumes a large enough initial dataset.

For each choice of a model space corresponds specific learning algorithms, that adjust the model to minimize the chosen loss. Several choices are possible, starting with linear regression, that only generalizes well if the data is linear, and polynomial regression, that has very poor generalization performances. We will focus in this thesis on three families of models: Decision trees and its variants, and the associated ensemble method, Random Forests, that pertain to symbolic AI; Neural Networks, subsymbolic approaches that have encountered incredible successes in the past 10 years, when handling huge amounts of examples; Genetic Programming, another symbolic approach based on artificial Darwinian evolution.

Random Forests are used as reduced models for all four algorithms of the first and second contributions of this thesis. Their implementation is defined in Section 4.4, then used throughout Chapters 4 and 5, which respectively tackle the first and second objectives of the thesis. Similarly, Neural Networks are used throughout Chapters 4 and 5 as substitution models of the simulator to deal with technical limitations that prevent a massive simulation plan, after being defined in Section 4.3. They are also considered as a baseline approach for border models in Section 6.2. Finally, Genetic Programming constitutes the final possible direction for building a border model as shown in Section 6.4.

3.2.1 Decision Trees and Random Forests

Decision Trees They are models for supervised learning (classification or regression), that are sets of if-then-else decision rules that can be represented as a tree (see Figure 3.1). Each terminal (aka leave) is associated with an output value (class for classification, real value for regression). For instance, in Figure 3.1, the leaves correspond to class *Good/No Good* which describes the failure state of the autonomous vehicle. Each node represents one rule that looks at one feature (aka attribute) of the current example, and descends the tree based on its value. For a real-valued attribute x_i, the condition has the form "$x_i < c$" for some real constant c while for categorical attributes, it has the form "x_i is c_1 or c_2 or ...".

The evaluation of an example (inference phase) starts with the root node, which is a decision node, and repeatedly checks the condition of the current decision node, and goes to either the right child if the condition is satisfied, or to the left child otherwise, until a terminal is reached. The tree then returns the output value associated with this terminal, a class for classification trees, a real-value for regression trees. In particular, the function represented by a regression tree is piecewise constant.

The learning phase recursively builds the tree, maintaining a subset of the training set at each node. The process starts with the whole training set at the training subset of the root

node. For each node, a decision has to be made whether to split the node or not, and if yes, according to which attribute, and with which values for this attribute. If the node is not split, it becomes a terminal node, and typically outputs the majority class of the current training subset for classification, or the average of the labels of all examples of the current training subset for regression. If the node is split, the attribute and the splitting value are decided according to some metric on the current training subset, and two child nodes are created, each inheriting the examples of the training subset of the parent node that satisfy the newly created condition. The process is iterated until reaching terminal nodes on all branches.

Whatever the metric used to split the nodes, and because the goal is to minimize the errors on the training set, it is clear that a perfect solution can always be obtained by going deep enough, until all terminals only contain a single training example. However, the tree will only have perfectly memorized the training set, and generalisation will be poor: this would be a typical case of overfitting. The depth of the trees must hence be limited, though shallow trees will have a very poor accuracy on the training set itself - not to mention the generalisation accuracy. The maximum tree depth is thus a crucial hyperparameter of Decision Trees.

We will now detail the metrics that can be used for the node splitting operation.

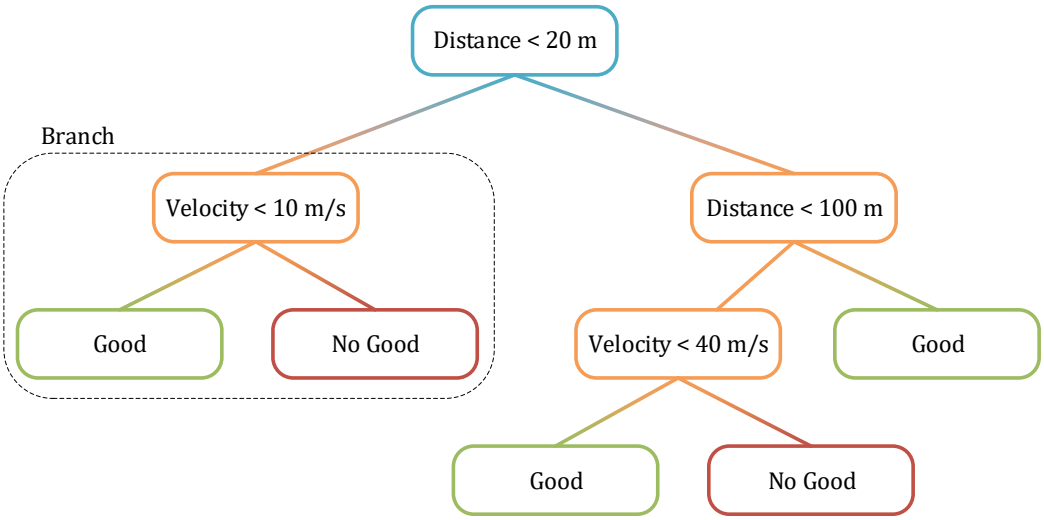

Figure 3.1: Visualization of a standard decision tree starting from the root node through the decision nodes leading to the terminal nodes.

ID3 The ID3 (Iterative Dichotomiser) algorithm (Quinlan, 1986) was originally designed to perform classification tasks. The node splits rely on information gain (IG) from information theory, measuring the change in entropy resulting from the split. The entropy at a given node

is classically defined by:

$$H = -\sum_{1}^{k} p_i log_2(p_i)$$

where p_i is the proportion of examples in class i in the training subset of the node. It measures the randomness of the current subset (0 if all examples belong to the same class, maximal if they are uniformly spread across all classes). The Information Gain for a given split is then defined by:

$$IG = H(before) - \sum_{j=1}^{K} H(j, after)$$

where $H(before)$ is the entropy of the current subset before the split, K is the number of new subsets generated by the split, and $H(j, after)$ is the entropy of the subset j after the split. The attribute that maximizes the IG is chosen for the split. Thus, an increase in information gain means a decrease in entropy. Alternatively stated, if a branch has zero entropy, meaning we are certain of the output, then it should be a leaf node, whereas another branch with entropy more than zero will be further split by ID3 until reaching zero – or the maximum depth allowed, or some minimal number of samples in the node subset.

For Regression trees, the Sum of Squared Error (SSE) is used in lieu of the entropy. For a given node, it is defined as:

$$SSE = \sum_{1}^{n} (y_i - \hat{y})^2$$

where y_i is the label of example i and $\hat{y}$ the average value of all labels of the training subset of the current node. The attribute and threshold that minimize the sum of SSEs of the child nodes after the split are chosen to actually perform the split. Here again, when a single example is left in a node, the SSE is 0, and overfitting most probably happens. So some maximal depth, or some minimal number of examples in the leaf nodes have to be enforced.

C4.5 Quinlan (1993) extended the ID3 algorithm and developed the C4.5 algorithm. In fact, the information gain metric is biased towards picking the attributes with a higher number of values (in the classification context). C4.5 works around this flaw by employing gain ratio which takes into account the resulting number of branches before making the split. In other words, the splitting metric used is a normalized information gain.

CART The CART (Classification And Regression Tree) algorithm builds a decision tree based on the Gini impurity index as splitting criterion (Breiman et al., 1984). While information gain is biased towards partitions with distinct values, the Gini index favors larger partitions and is easier to implement. It can be calculated by summing the probability p_i of a picked item labelled i, then multiply this sum with the probability $\sum_{k \neq i} p_k = 1 - p_i$ of a mistake in categorizing that item. If it reaches its minimum, which is zero, it means that all samples in the node fall into a single target category, turning the node into a leaf.

As for regression applications, SSEs are again used to split points, and overfitting still most probably happens.

Random forests In a nutshell, a random forest is an ensemble method that combines multiple decision trees. In case of classification, it outputs the majority vote of its trees, while in the case of regression, it outputs the mean prediction of the individual trees.

Random forests (Breiman, 2001) were originally conceived as an algorithm that combines several CART decision trees using the generic technique of bagging (Breiman, 1996), which creates an ensemble of classifiers from multiple training sets that are sampled with replacement from the original one. More randomness was added by only considering a random subset of the attributes when training each tree. It has been demonstrated (Biau, 2012) that substantial gains in classification or regression accuracy can be achieved by using such ensembles of trees. Random Forests are fast, both at training and inference time, accurate when well trained, and indeed very robust to fight overfitting when a high number of trees is used.

Discussion The main advantages of decision trees are their ease of use, as the if-else statements give a clear visualization easily interpreted and understood, and their broad application ranges for both classification and regression problems, as well as handling both continuous and categorical variables. They also usually take a less training period than more sophisticated algorithms such as the neural networks seen in the next section. However, the main problem of the decision trees is the overfitting of the data which ultimately leads to false predictions on the unseen data, as the tree loses its generalization capabilities while trying to fit all the data into new nodes. This problem leads to instability of the decision trees as they are highly affected by noisy data. Therefore, the random forests are seen as the best version of decision trees (Ali et al., 2012) thanks to the bagging method that addresses the overfitting problem and its underlying limitations, while maintaining all the main advantages of the decision trees.

3.2.2 Deep Neural Networks

Though first proposed in the 50's (Rosenblatt, 1958), neural networks became utterly successful only about a decade ago, thanks to i) the availability of huge amounts of data; ii) the amazing growth of the available computing power; and iii) new optimization algorithms, efficient variants of classical gradient algorithms. This allowed to be able to train what is now called Deep Neural Networks (DNNs), considered the main recent breakthrough in Machine Learning with numerous impressive results (Krizhevsky et al., 2012; Goodfellow et al., 2014). Their easy portability to Graphical Processing Units (GPUs) (Raina et al., 2009) thanks to modern Open Source libraries like TensorFlow and PyTorch made it possible to handle huge datasets and was the source of their breakthrough performance in many domains (from

computer vision to Natural Language Processing, to name the most prominent), and hence of their ever increasing popularity.

In a nutshell, a typical (feedforward) neural network[1] consists of multiple layers of elementary computing units, aka *artificial neurons*, as they grossly resemble biological neurons. Each unit computes a non-linear function (the *activation function*) of the weighted sums of its inputs. The outputs of layer N are the inputs of layer N+1. The inputs of the first layer are the features describing the training examples, and the outputs of the last layer are the output of the model. The learning phase consists in tuning the weights in order to minimize a *loss function* that measures the error of the network (see below). *Stochastic gradient descent* is the optimization method of choice to reach a local minimum of the non-convex loss function, even though other methods have been used in the early days of NN history. We will now give more details on the important issues when using (deep) neural networks for regression, starting with the historical Perceptron.

The Perceptron The simplest architecture of a NN is when it contains only one neuron. This model is named the Perceptron (Rosenblatt, 1958). It is a linear classifier, from which stem all recent artificial neural network architectures. If we consider an input $\mathbf{x}$ of dimension d, the output $\widehat{\mathbf{y}}$ of a Perceptron is:

$$\widehat{\mathbf{y}} = \sigma(\mathbf{w} \cdot \mathbf{x})$$

where $\mathbf{w}$ is the vector of *weights* (by convention, $\mathbf{x}$ is extended by a constant input $x_0 = 1$, and w_0 is also called the bias), σ is the Heaviside function. In supervised machine learning, the training data is made of examples of input-output pairs, and the aim is to learn the best weights W that minimize the chosen loss function (see below), i.e., get $\widehat{\mathbf{y}}$ to be as close as possible to the real labels $\mathbf{y}$ for all $\mathbf{x}$ in the training data.

Artificial neural networks are biologically-inspired, and the first learning rules took inspiration from the Hebbian theory, that claims that when a presynaptic cell stimulates repeatedly and persistently a postsynaptic cell, the synaptic efficacy increases (Hebb, 1949). Therefore, there should be proportionality between the weights updates $\mathbf{w}_i$ and the correlation between input x_i and output $\mathbf{y}$. One example of algorithm that implements this proportion is the Widrow-Hoff algorithm, with update rule $w_i' = w_i + \alpha * (\mathbf{y} - \widehat{\mathbf{y}})x_i$ (despite not incorporating an activation function). In short, the idea is to update the weights using sufficient data in order to output a model that will figure out how to pass from input $\mathbf{x}$ to an output $\mathbf{y}$.

Nonetheless, the key limitation of the linear Perceptron is its inability in dealing with inputs that are not linearly separable, such as in the XOR problem. This led to the creation of the multi-layer perceptrons in the end of the 80s, the first non-linear neural network architectures.

Multi-layer perceptrons The idea is to stack several layers of neurons, where each one of them is defined by its own learnable weights, and a non-linear activation function. A typical

[1] Many other more complex architectures exist, that will not be touched upon here.

multi-layer architecture, called fully connected NN, is illustrated in Figure 3.2: All neurons of layer i are connected to all neurons of layer $i+1$. But many other partial connection graphs have been proposed and used, the only requirement is to avoid loops in the connectivity, allowing to compute the activation values from the inputs to the outputs in a single pass: this characterizes the class of *feedforward* NNs.

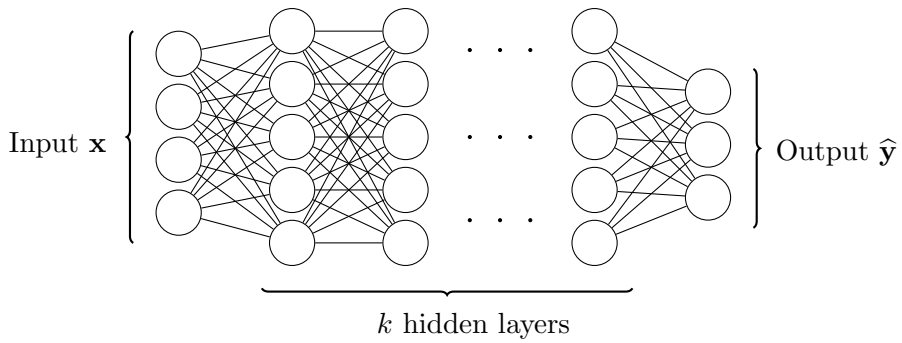

Figure 3.2: A multi-layer perceptron. The first layer is the input layer $\mathbf{x}$, whereas the last layer is the output layer $\widehat{\mathbf{y}}$. All the k layers in between are called *hidden layers*, and each node and edge represent respectively a neuron with its activation function, and a learnable weight.

If the NN consists of an input layer, an output layer, and only a few (typically one) hidden layers, the NN is called today "shallow", whereas if it contains many hidden layers, the NN qualifies as "deep". Formally, the output $\widehat{\mathbf{y}}$ of a multi-layer perceptron with input $\mathbf{x}$ and k layers are given by the following equation:

$$\begin{cases} f_i(\mathbf{x}) = (\sigma_i(\mathbf{w}_i \cdot \mathbf{x})) \\ \widehat{\mathbf{y}} = f_k \circ f_{k-1} \circ \ldots \circ f_1(\mathbf{x}) \end{cases} \tag{3.3}$$

We notice that the NN architecture is versatile w.r.t. the dimensions of the inputs and outputs, i.e., the dimensions of the weight vectors $\mathbf{w}_i$ can be easily modified to adjust to the available data, and to any feedforward architecture.

Activation functions As explained in the previous paragraph, σ_i is the non-linear activation function that introduces the non-linearity in the NN architecture, as all other operations between inputs and the weights are linear. The need for having non-linear activation functions in the first place is because, if we use linear activation functions, the predicted output is simply a linear approximation of the output, no matter how many layers in the architecture – and non-linearly separable problems simply cannot be solved.

Examples of common activation functions are the Sigmoid function, the Hyperbolic Tangent (tanh) and the Rectified Linear Unit (ReLU), Sigmoid and ReLu being the ones used

throughout this thesis. All three functions are continuous increasing functions, as illustrated in Figure 3.3.

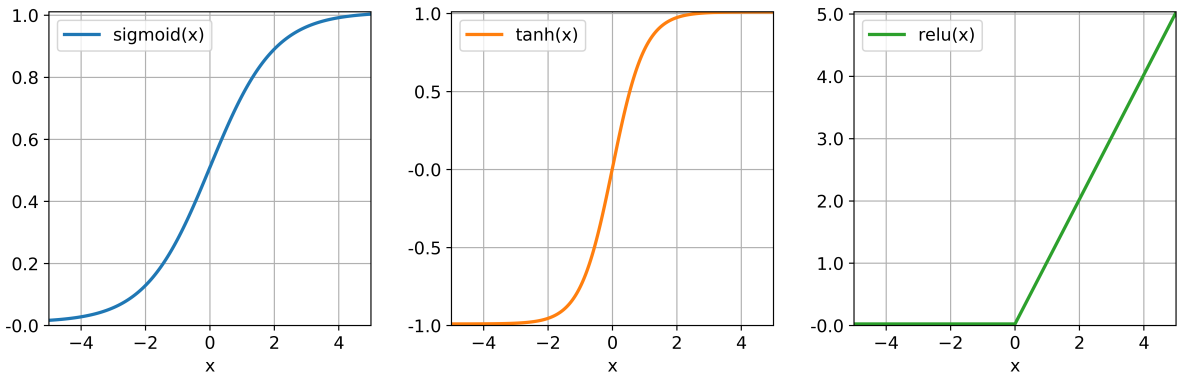

Figure 3.3: Illustrations of non-linear activation functions examples used in the implementation of neural networks.

- **Sigmoid** is bounded between 0 and 1. It is essentially used for binary classification problems, because it can convert any value to a 0 or 1 probability for two classifications when implemented in the output layer. It converges to the Heaviside function when the slope of the derivative at 0 goes to infinity.

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

- **Hyperbolic Tangent (tanh)** is bounded between -1 and 1, centered at 0 which makes it suitable for centered data sets and most other cases than binary classification. However, it is considered to be computationally expensive compared to alternative activation functions.

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 = \frac{e^{2x} - 1}{e^{2x} + 1}$$

- **Rectified Linear Unit (ReLU)** is the most popular choice nowadays. It is not bounded nor fully differentiable, but it is convex, and its (sub-)gradient is 0 if the input is negative. However, it allows quick computation because of the simplicity of the function and its derivative.

$$ReLu(x) = Max(0, x)$$

Losses In order to learn the weights of a feedforward neural network in the context of supervised learning, a loss function must be defined, suitable to the problem at hand. This function should denote how close the predictions provided by the neural network are to the expected predictions, i.e., the actual labels. Two popular losses are used throughout this thesis, and are detailed next.

- **Mean-Squared Error:** This loss is typical for regression problems because of its stability and robustness. The equation of such loss is:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \widehat{y}_i)^2 \tag{3.4}$$

  where m accounts for the number of examples in the training set, y_i the label of the ith sample, and $\widehat{y}_i$ the network prediction of the ith sample.

- **Cross-Entropy:** This loss is standard for classification problems. While it is computed using entropies, we do not have access to the probability distributions of the labels and predictions in practical applications. Therefore, an approximate equation of the loss is defined with the respective sets of samples $\mathbf{y}$ and $\widehat{\mathbf{y}}$. In the case of binary classification, with only two classes, this gives the following loss:

$$CE = -\frac{1}{m}\sum_{i=1}^{m}\Big(y_i log(\widehat{y}_i) + (1 - y_i) log(1 - \widehat{y}_i)\Big) \tag{3.5}$$

Stochastic Gradient Descent After choosing the loss function, the goal is to globally minimize it. It turns out that the output is a (sub-)differentiable function of the weights and the inputs. Hence if the loss function is also differentiable, it is possible to compute the derivatives of the loss function w.r.t. the weights, and from there on, to use any gradient method to minimize the loss. Considering the huge size of the training set, the preferred method is the Stochastic Gradient Descent (SGD), that uses only a subset of the whole training set (aka *mini-batch*) to compute a partial gradient and update the weights accordingly. Furthermore, thanks to the chain rule of computation of derivatives, it is possible, for each example of the training set, to first compute the outputs of the network (because of the absence of loops in the topology of the network) starting from the input layer (feedforward propagation), and then to compute the derivatives w.r.t. the different weights by back-propagating the error from the last layer (for this reason, this algorithm is called "gradient backpropagation"). The weight update is then computed using the standard gradient update:

$$\mathbf{w}_i \leftarrow \mathbf{w}_i - \eta\frac{1}{m}\sum_{j=1}^{m}\nabla\mathcal{L}_j(\mathbf{w}_i) \tag{3.6}$$

where $\mathcal{L}_j$ is the value of the loss $\mathcal{L}$ for the jth example of the mini-batch, and η is *the learning rate*, an important and influencial hyper-parameter.

Training the neural network The basic training phase proceeds iteratively: First, the training set is divided into several mini-batches, and for each mini-batch, the corresponding examples are presented to the network one by one. For each example, forward propagation predicts the corresponding outputs, the loss function is computed, and back-propagation is run to compute the gradients for each weight of the network. At the end of the mini-batch,

Algorithm 1: Neural network learning procedure (Stochastic Gradient Descent)

```
for number of epochs do
    for number of mini-batches do
        • Sample a batch x_i out of the training data set
        • Predict the output ŷ_i
        • Calculate the loss w.r.t the real labels y_i: L_i = L(ŷ_i, y_i)
        • Compute the gradients for all the learnable parameters ∇L_i(w_k)
        • Update the learnable weights according to Equation (3.6)
    end
end
```

the weights are updated using Equation (3.6). An *epoch* refers to processing the whole set of mini-batches (i.e., every example of the training set once). The training phase of a NN is summarized in Algorithm 1.

Improving the basic training algorithm Numerous improvements of the basic algorithm described above have been proposed, and describing them all is beyond the goal of this work. However, some of the most efficient (and hence popular) are used throughout this thesis, and we will now briefly introduce them.

- **Learning rate and optimizer:** The learning rate of the stochastic gradient descent, as defined in Equation 3.6, is an important hyper-parameter to be effectively tuned. It defines the speed of adaptation of the weights. If its value is too high, the optimization might crash on some gradient barriers, and the gradient will diverge, or not be able to reach any local minimum, overshooting it again and again without ever being able to move with the correct precision. If it is too low, escaping local minima becomes difficult, and in any case, even reaching a local minimum might take an enormous number of iterations. Thus, the impact of the learning rate on the neural network is quite significant.

  One way of dealing with this caveat is to apply momentum-based optimization which will enable faster learning using exponentially weighted averages. It allows the gradient descent to gain more context on the optimization process and adapt its learning rate accordingly. Another popular optimizer that works well in a lot of applications is called **Adam** (Kingma and Ba, 2014). Instead of shifting the weights values with the gradient mean value in a linear fashion, it takes into consideration gradient statistics of higher

order, using gradients decaying averages m and squared gradients v as follows:

$$\begin{cases} m(\mathbf{w}_i) \leftarrow \beta_1 m(\mathbf{w}_i) + (1-\beta_1)\nabla\mathcal{L}(\mathbf{w}_i) \\ v(\mathbf{w}_i) \leftarrow \beta_2 v(\mathbf{w}_i) + (1-\beta_2)\nabla\mathcal{L}(\mathbf{w}_i)^2 \\ \hat{m}(\mathbf{w}_i) \leftarrow m(\mathbf{w}_i)/(1-\beta_1) \\ \hat{v}(\mathbf{w}_i) \leftarrow v(\mathbf{w}_i)/(1-\beta_2) \end{cases} \tag{3.7}$$

$$\mathbf{w}_i \leftarrow \mathbf{w}_i - \frac{\eta}{\sqrt{\hat{v}(\mathbf{w}_i)} + \epsilon}\,\hat{m}(\mathbf{w}_i) \tag{3.8}$$

where β_1 and β_2 are two hyperparameters that control the decay rate. But Adam seems pretty robust w.r.t. these hyper-parameters, that are generally taken as recommended by Kingma and Ba (2014) – with the exception of the initial learning rate η_0 that still needs to be manually tuned.

- **Weights initialization:** The initialization of the weights at the beginning of the training process of the neural network also has a strong impact on the performance of the training. For instance, it is not a good idea to initialize all weights with zeros, or set them at a given constant value, because all gradients will be the same across the network, resulting in identical weights all along. On the other hand, if the initial weights are too high, all neurons will be saturated, and return a value close to the upper or lower bound, with almost null gradient: the weights will not evolve.

  Thus, many heuristic approaches have been proposed to aid the network in training properly, such as random initialization using a Gaussian or uniform distribution. The most popular heuristic to date is that of Xavier or He (Glorot and Bengio, 2010; He et al., 2015), and was used throughout this thesis: the idea is to keep the variance fixed from layer to layer in both the feedforward and back-propagation directions, so as to not saturate the neurons and enable the network to learn optimally. These respective variances are set as follows:

$$Xavier : Var(\mathbf{w}_i) = \frac{2}{size^{[l-1]} + size^{[l]}}$$

$$He : Var(\mathbf{w}_i) = \frac{2}{size^{[l-1]}}$$

  where $size^{[l-1]}$ and $size^{[l]}$ represent the number of incoming and outgoing connections respectively.

- **Regularization:** A trained neural network should give accurate predictions on its training set. If this is not the case, it means that it is underfitting training data. Some solutions that could fix this problem are to: train the network longer; have a bigger network; or change the NN architecture. On the other hand, overfitting occurs when training the network longer leads to a decrease of the error on the training data, but

ultimately increases that on unseen data (the test set): the network had been trained to perfectly "memorize" the training data, regardless of any generalization ability.

In order to handle this problem, regularization is used by adding constraints to the loss which limits the ability of the model. Adding the L_1 or/and L_2 norm of the weights to the loss is a standard way to regularize. However, more NN-specific approaches have been proposed, such as the dropout (Srivastava et al., 2014). It consists in randomly cutting off some neural connections with a given probability during each epoch of the training phase. This method ensures that no co-adaptation occurs between all the network weights, and this can efficiently prevent overfitting.

Hyper-parameters As described above, before running a deep neural network for some supervised learning task, there are a lot of parameters that should be fixed by the user, after the problem (inputs, outputs, loss function) has been defined. The first ones are those of the architecture: number of hidden layers, type of each layer (e.g., fully connected, convolutional, pooling, residual, ...), their sizes (number of neurons), and the type of activation function. But there are also numerous other hyper-parameters, i.e., parameters that are chosen at the conception of the model, and are usually not updated during the training phase: choice of the optimizer, that comes with its own hyper-parameters, learning rate, and in particular its initial value and how it decays during the iterations, dropout or not dropout (Srivastava et al., 2014), stopping criterion (e.g., early stopping), to name the most prominent ones. Like for all other Machine Learning approaches, these hyper-parameters are usually adjusted using an independent validation dataset. This hyper-parameter search ranges from random search (Bergstra and Bengio, 2012) to grid search to more sophisticated search methods that belong to Automatic Machine Learning (AutoML) branch of research (Feurer et al., 2015; Golovin et al., 2017; Gordon et al., 2018). The quest for the architecture alone is today called Neural Architecture Search (NAS) (Elsken et al., 2019; Wistuba et al., 2019).

Discussion Though Deep Neural Networks led to incredible progresses in many application domains, they (still) suffer from a number of drawbacks. First of all, they require a large amount of training data, and not all domains have hundreds of thousands of examples at hand. Techniques like transfer learning and domain adaptation somehow mitigate this, but this remains a prerequisite in most applications. DNNs also suffer from a lack of explainability, a hot topic in today's research: the millions of weights of a learned model can hardly be interpreted, and though new approaches are proposed every day, none stands out generic enough. Finally, DNNs lack robustness in front of adversarial attacks, or when facing unseen noise (e.g., photos in different lightnings). And though many works start to address the certification of DNNs using formal methods, it seems that the goal is still rather far away.

3.2.3 Genetic Programming

Genetic Programming (GP) (Banzhaf et al., 1998) is a branch of Evolutionary Algorithms that aims at synthetizing programs. The historical form of GP, and still the most prominent, represents programs as trees: for instance, Koza (1992) used a subset of the LISP language, and his goal was "to let the computer write the program that solves the problem" by simply providing a loss function to minimize. The optimization is performed using an Evolutionary Algorithm, i.e., a stochastic optimization algorithm mimicking Darwinian evolution. Note however that many different representations of programs have been proposed: linear GP (Brameier and Banzhaf, 2010) handles sequences of instructions; Cartesian GP (Miller, 2020) sets the operators and terminals on a grid; stack-based GP (Perkis, 1994) uses operators of a stack-based programming languages; Grammatical Evolution (Connor et al., 2019) uses a Context-Free Grammar to decode sequences of codons into a program. Nonetheless, they will not be mentioned any further.

In the context of this work, tree-based GP will be used as a *symbolic regression* tool: trees represent real-valued functions of real-valued variables. In this respect, the semantics of the language represented by GP trees is much richer than that of Decision Trees, which is limited to if-then-else rules with a single condition on one attribute of the data.

Representation Given a problem defined on some space $\mathcal{S}$, given a set of nodes $\mathcal{N}$, or operators acting on elements of $\mathcal{S}$, and a set of terminals $\mathcal{T}$, usually the variables of the problem at hand defined on $\mathcal{S}$, trees are defined recursively: starting from a root operator node, all operator nodes are expanded into as many children as their arity requires: each child is either another operator node, or a terminal. The tree is complete when all branches end up with terminal nodes. The execution of the program represented by a given tree on an instance of the problem (a vector of $\mathcal{S}^n$) starts by instantiating all terminals, and recursively computes the values returned by the operators up the tree. The program returns the value computed at the root node.

In this framework, Boolean functions of logical variables $x_1, \dots x_n$ can be represented using the binary elementary logical functions ($\mathcal{N} = \{\text{AND, OR, NOT}\}$) and the variables of the problems as terminals ($\mathcal{T} = \{x_1, \dots x_n\}$); Real-valued function of real-valued variables $x_1, \dots x_n$ can be represented by trees using binary elementary operations ($\mathcal{N} = \{+, -, *, /\}$)[2] and variables and constants as terminals ($\mathcal{T} = \{x_1, \dots x_n, \mathcal{R}\}$) where $\mathcal{R}$ represents real-valued constants, aka *ephemeral constants*, real values that are defined at initialization time and are not modified any more.

For instance, the following small algebraic term relating two inputs x_1 and x_2:

$$x_1^3 - (x_2 - 2) \tag{3.9}$$

[2]the division is in fact the protected division, that returns 1 if the denominator is 0

can be represented in LISP-like language as:

$$(-(*x_1(*x_1x_1))(-x_22)) \tag{3.10}$$

which, in turn, is represented by the tree in Figure 3.4.

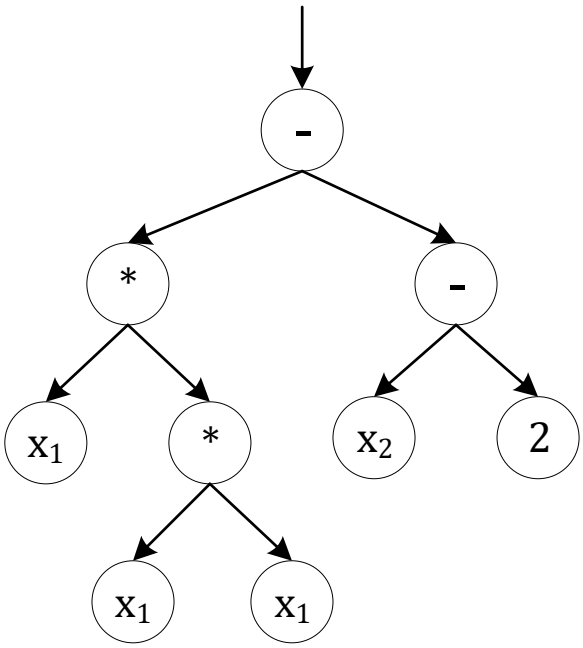

Figure 3.4: Tree representation of the LISP format of Equation 3.10.

Much more complex set of nodes can be used, in particular containing operators of different arities. For instance, in the context of real-valued functions of real-valued variables, $\mathcal{N}$ might contain unary trigonometric functions (sine, cosine, tangent, ...), logarithms and exponentials, and hyperbolic trigonometric functions, or any pre-defined real-valued function. But $\mathcal{N}$ might also include the *if-then-else* ternary function that takes a condition as first child (a full subtree, returning some real value), and executes the second child if the result of the first child is positive, and the third child otherwise.

Evolutionary Algorithms Evolutionary Algorithms (EAs) are bio-inspired stochastic optimization algorithms. They are based on the basic principles of Darwinian evolution, coupling *Survival of the fittest* and *blind variations*. As such, the field of EAs uses a specific vocabulary inspired by its biological model.

Given a real-valued objective function defined on a search space Ω, to be maximized (aka

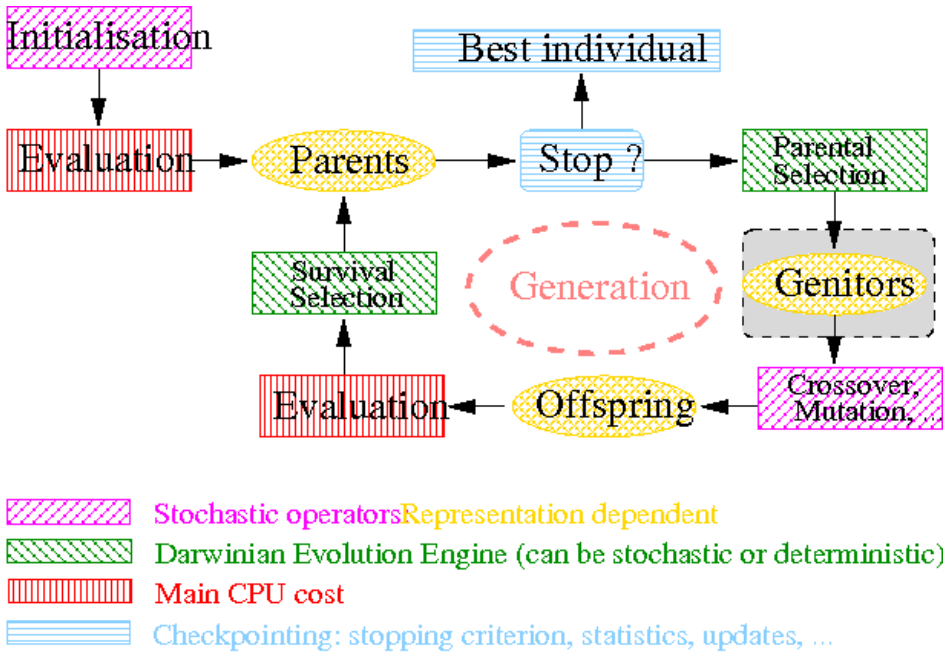

Figure 3.5: Basic Evolutionary Algorithm.

fitness function), a basic EA evolves a set of points of Ω (aka a *population* of P *individuals*), during a succession of iterations (aka *generations*). The population is initialized randomly, the fitness of all individuals is computed. At each generation, the current population (aka the *parents*) undergoes *parental selection* to select the *genitors* based on their fitness (first implementation of the *Darwinian selection*). They undergo some random perturbations (aka *variation operators*, grouped in two families, *crossover* and *mutation*) that generate new points of Ω (aka *offspring*). These offspring are in turn evaluated (their fitness is computed), and a final Darwinian *survival selection* reduces the set of parents plus offspring to size P. Figure 3.5 gives a synthetic view of such basic EA.

The first historical EAs are Genetic Algorithms (Goldberg, 1989) that work on bitstring ($\Omega = \{0,1\}^N$), but the strength of EAs comes from their ability to handle any "weird" search space Ω, provided you can define blind variation operators. Note that the word "blind" here means that these operators are not related to the fitness, that is taken care of by the Darwinian selection part of the algorithm. It is clear, however, that the key to success for EAs is that the semantic of these variation operators is somehow related to the objective of the optimization problem at hand.

In particular, EAs apply to search spaces where no "classical" optimization algorithm can be easily applied ... like the space of programs defined by parse trees. Nevertheless, EAs have obtained many successes in various application domains, as witnessed for instance by the list of recipients of the *Humies Awards* (`http://www.human-competitive.org/awards`).

Let us briefly describe the representation-independent components of an EA, before de-

tailing the representation-dependent parts for tree-based Genetic Programming used in this thesis.

Darwininan Selection Historically, the first selection method used in GAs (Goldberg, 1989) is the so-called roulette-wheel selection: each individual is selected with a probability proportional to its fitness (the goal is to maximize the fitness). However, this selection mechanism suffers many defects, starting with its sensitivity to the scaling of the fitness. It is hence abandoned nowadays, and the preferred selection method of modern EAs is the tournament selection. In tournament of size T, in order to select one individual, T individuals are uniformly selected from the population, and the best individual out of these T is returned. The tournament size T influences the selection strength: high values will mainly select the best individuals from the population, while low values (e.g., 2) will also select some low-fitness individuals, hence favoring a greater diversity in the population.

Representation-Dependent Components As described in Figure 3.5, variation operators as well as initialisation depend on the chosen representation. This Section will present the ones mostly used for tree-based GP.

There are basically two types of variation operators: *Mutation operators* are unary operators that randomly modify an individual (i.e., transform it into another point of Ω); *Crossover operators* are binary operators that generate children (new points of Ω) from the two parents.

The underlying idea behind crossover operators is that they should allow "good parts" of both parents (aka *"building blocks"*) to be recombined and possibly lead to children that perform better than both parents. Mutation operators might have two roles: making large moves to escape local optima, or allowing fine tuning when nearing the optimal solution.

Finally, initialization should allow to start with a good coverage of the search space – ideally a uniform coverage. But such uniform distribution doesn't exist on non-standard search spaces like that of GP trees.

Tree crossover Probably the main source of power of GP is the crossover operator, that exchanges two subtrees of the parent trees (see Figure 3.6 for an illustration). Thanks to the tree representation, all trees are valid programs (or functions), hence the crossover operator is closed in the space of tree-based programs. In order to try to control the disruption brought by the crossover, it is possible to choose the crossover points (location of the subtrees to be swapped) depending on their depth in the tree, deep crossover point being less prone to high disruption than higher ones.

Also, in order to better match the semantic of the problem of symbolic regression (see below), some variants of this basic crossover have been proposed, like the semantic crossover, in which the actual values taken by the subtrees on the fitness cases are used.

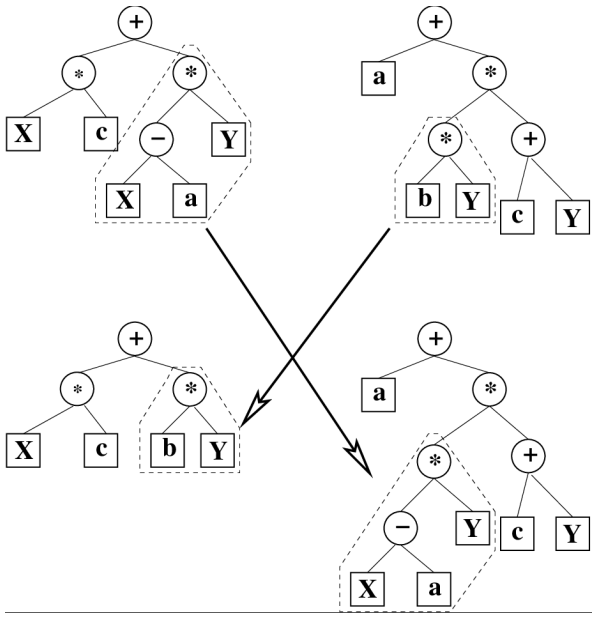

Figure 3.6: Example of tree crossover.

Tree Mutations Several mutation operators are available. Point mutation randomly cuts a subtree and replaces it with a completely new tree (see Initialisation). Node mutation randomly selects one node and switches for another operator. Similarly, terminal mutation replaces one terminal with another terminal (including ephemeral constants). A specific constant mutation modifies the value of ephemeral constants. More complex mutation can also be used, like insertion and deletion of subtrees, semantic variants, etc

Tree Initialization A naive initialization procedure proceeds from the root node (uniformly chosen among the different operators), and recursively goes down the tree, choosing uniformly among operators and terminals, stopping when only terminals remain on the stack (this procedure is called *Grow*). It is clear, however, that such procedure could generate very deep trees, as this process is unbounded. It is hence necessary to set a maximal depth. On the other hand, the procedure described above can also generate very short trees. One way to get trees of a desired depth d is to choose only nodes during the development of the tree until reaching depth d and to choose only among terminals there - procedure termed *Full*.

The most popular initialization procedure to date uses both Grow and Full, and is called *Ramped Half and Half*: given a maximum depth D and a population size P, the idea is to generate, for each depth $d \leq D$ exactly $P/2D$ trees of maximum depth d with procedure Grow, and $P/2D$ trees of depth d with procedure Full.

Fitness Cases and Fitness Functions In the context of symbolic regression, the training set is made of examples $(\mathbf{x}_i, \mathbf{y}_i)$ (see Section 3.2). In GP, each example is also named *fitness*

case. And the fitness is, in most cases, the Mean Squared Error (MSE) loss function defined in Equation (3.4). However, different terms can be added to the MSE, in order to favor different characteristics of the solution.

One of the main issue with standard GP is the so-called *bloat* phenomenon, i.e., the quick increase in size of the GP trees across the generations. Bloat not only increases the computational consumption of the whole process, it also decreases the explainability of the solution, despite the analytic form obtained. Multiple extensions have been proposed to fight bloat, like limiting the tree depth or applying operators with depth-dependant crossover (Ito et al., 1998), while others attempt to deal with the impact of endlessly changing operators by using subroutines detached from the main body of a code (Koza, 1993; Angeline and Pollack, 1997). One popular approach is to add in the fitness a regularization penalty depending on the size of the tree.

Discussion While tree-based GP can be used as a symbolic regression tool to represent real-valued functions in a way that is richer than that of Decision Trees with their limited if-then-else rules, the evolutionary algorithm employed requires multiple parameters to be tuned, e.g., population size, number of generations, tournament size, crossover and mutation probabilities... Plus, the bloat phenomenon remains the main issue of standard GP which increases computational cost and decreases solution explainability. Nonetheless, the key advantage of EAs is their broad application range as they are able to tackle any search space if all necessary operators and conditions are correctly defined.

Chapter 4

Methodology and Experimental Conditions

This Chapter introduces the general context of the optimization algorithms developed in this thesis, and illustrates the methodology with results regarding the simulation-based identification of failures in the decision system (first objective of the thesis).

4.1 Problem statement

The main goal of this work is to validate the command law of a decision system for an autonomous vehicle for a given use case using a simulator, i.e., to identify the region of the space of all possible scenarios for the use case at hand where the command law fails – or to bound the size of this region. But let us first give more precise definitions of the different concepts at play here.

4.1.1 Use cases and Scenarios

As explained in Section 1.4.3, all driving sequences of the autonomous vehicle can be enumerated, as far as simulation is concerned, as hundreds of *use cases*. Each use case is a description of several vehicles in a given physical environment: number of vehicles (in addition of the car under study, also called *EGO* car) that will be circulating, description of the road segments and trajectories of the different vehicles (including the EGO car), etc

Furthermore, each use case is characterized by the list of input variables of the simulator that characterize the driving situation of the autonomous vehicle. These input variables are exterior to the EGO vehicle, and can include road conditions, weather variables, etc. The number of such input variables is called the dimension of the use case. Setting the values of all input variables defines a *scenario*, which is an instanciation of a use case. Thus, the space of possible scenarios for a use case has as dimension the use case dimension, and is defined by the range or list of values of each input parameter. Therefore, the environment

of the autonomous vehicle can be defined accurately in order to check the behavior of the autonomous vehicle command law in different situations.

The simulator aims at reproducing as accurately as possible the evolution of all the cars of a scenario: the EGO car is driven by the command law under test, the other vehicles follow the trajectories of the use case using the parameters of the scenario. In particular, the simulator computes different indicators of the driving conditions of the EGO car, and outputs some statistics based on these indicators, called *warnings*. They are defined with the goal of identifying the autonomous vehicle behavior under multiple aspects of interest following a given simulated driving situation. For instance, these warnings are organized by theme, e.g. safety, testing, and marketability.

For each of the warnings, a range of values is considered dangerous, and when falling into these values, the criterion is then said to be *No Good*, abbreviated as *NG* in the following. On the other hand, all other values, corresponding to safe behaviors, are qualified as *Good*, written *G*. Note that one single NG criterion qualifies the whole scenario as *No Good*, but we will nevertheless work criterion by criterion in the rest of this thesis.

4.1.2 The optimization algorithms

For a given use case, we would like to identify sets of input values of the simulator that result in specific properties of the outputs: either scenarios raising a failure of the command law, or pairs of scenarios to help identify the border of the failure zone. One way to detect such scenarios could be to discretize all continuous parameters of the scenarios at hand, and perform an exhaustive grid search on these input parameters. However, such approach is clearly bound to combinatorial explosion: For the simple use case described in Section 4.5.1, assuming we take into account all the values of the two discrete inputs of the tracking vehicle, and discretize all five continuous inputs after normalization between 0 and 1 with step size of 0.1, a rather coarse discretization, the resulting number of scenarios to be tested is nevertheless 3,865,224, which is huge for a very simple use case. Furthermore, such validation process needs to be executed for all use cases, and should be repeated every time the decision system is updated and modified during that whole process. Therefore, developing an optimization algorithm that performs only a handful of critical scenario simulations is mandatory for validating the milestones of Renault's industrial project ADValue while meeting the time and cost tight constraints.

4.2 Simulator and Surrogate Models

The simulator that was available during this thesis is *SCANeR Studio* (AVSimulation, 2017). It is an in-house software dedicated to automotive simulation including both driving and testing of ADAS and autonomous vehicle system. It combines numerous tools and models to provide realism as much as possible to the virtual simulation, e.g., road environment, traffic, vehicle dynamics, weather conditions... The desired scenario can be scripted to simulate a

certain driving configuration with the autonomous vehicle. However, the main disadvantage of the software is that we are obliged to wait for the whole simulation to conclude in order to collect the wanted outputs. For the simple use case described in 4.5.1, the scenario can take up to 5 minutes in simulation time. Plus, a software license should be bought for each terminal to be able to launch a simulation on it. Thus, running simulations in a parallel mode was impractical, at least at the beginning of the thesis, and all tests had to be run sequentially for the whole duration of the simulations. Note that the primordial requirement of the ADValue project is to minimize the number of simulations to be launched to reduce computational costs.

In order to cope with such expensive simulator, two key models will be created and used in the optimization loop of all developed algorithms for the failures identification and border detection. The *reduced model* is crucial in reducing the number of expensive simulations used in the optimization loop, while the *substitution model* is used because of the technical limitations which limit the possibility of a massive simulation plan. Nonetheless, it is important to not be confused between the two "surrogate" models. The substitution model is built using offline calculations from the current version of the simulation software. It is then considered the ground truth and is never updated once built. It replaces the simulation software, hence the name. On the other hand, the reduced model is used intensively in the optimization loop, and is continuously updated online during the optimization. Because of these different characteristics and usage, we have chosen two different types of models for these surrogate models: Neural Networks for the substitution model, and Random Forests for the reduced model (see Chapter 3).

4.3 The Substitution Models

During the course of the thesis, each simulation is very costly even for the simplest use cases (see Section 4.5.1), because we have to wait for the whole scenario to be simulated by the software, which takes minutes of simulation time. Furthermore, technical software limitations occurred which made a massive simulation plan impossible to be run in parallel. In fact, a license is required to be bought on each terminal to be able to use the simulation software. Plus, a massive simulation platform with tens of computing cores embedded with simulator licenses is currently being developed at Renault to provide the possibility of a massive simulation plan in parallel. However, at the beginning of the thesis, this platform was unavailable, and all tests had to be run sequentially.

Therefore, because of the high amount of simulations required to explore the scenario search space, hundreds of hours would have been needed to run a complete single test for a given use case, even if using a surrogate reduced model to approximate the true objective function. This situation prevented any real-size experiment involving development and testing of the algorithm. Hence, we decided to build, once and for all before all experiments of a use case, a *substitution model* of the simulation software, which can almost instantly output

the criteria of the scenario instead of running the costly simulations. After this substitution model has been built, its outputs are considered to be the only ground truth, and it totally replaces the actual simulation. In that way, the optimization algorithm can be designed and tested in a reasonable amount of time for this thesis, as multiple 'simulations' can then be run in parallel. However, it should be clear that this substitution model is just a temporary fix used for this thesis, and should be replaced by the real simulator as soon as possible, and in the long run when the massive simulation platform is fully ready. On the other hand, the reduced model is a center building block of the optimization loop, built anew in every experiment described in this thesis.

4.3.1 Deep Neural Networks as Substitution Models

The goal of the substitution model is to replace the actual simulation for the technical reasons given above during the different optimization algorithms. As such, it should be as accurate as possible so the lessons learned using this substitution model can be transferred to the real situation where the actual simulation is used. This requires using many examples for training the substitution model. Furthermore, the training time of the substitution model is not an issue here: gathering the examples is anyway very long, and a few more hours will not change the overall cost. On the other hand, the test of different scenarios with the substitution model should be quick enough to be included in some optimization loop without hampering the optimization itself. This is why Deep Neural Networks have been chosen as the base model for the substitution models of all use cases in this thesis.

Because the data has no particular structure, fully connected models have been chosen, with ReLU activation functions on all layers except for the output layer where we keep it linear for this regression problem. Tensorflow (Abadi et al., 2015) was used via its Python front-end to build and train these neural networks. Adam optimizer (Kingma and Ba, 2014) has been used in all experiments, and the two popular initialization procedures of the weights (Xavier and He) described in Section 3.2.2 have been tried. Mini-batch gradient descent optimization has also been used while testing batch sizes of 64, 128 and 256.

4.3.2 Building the Substitution Models

For the aim of building a reliable substitution model for a given use case, a design of experiments with a maximin decision rule (Johnson et al., 1990) is first run in the scenario space. Its goal is to create a quasi-random sequence of scenarios, so that the experimental design is composed of scenarios as far apart from one another as possible in the input parameter space. The size of this DoE depends on the dimension of the use case, i.e., the dimension of the scenario space. Typically, for the use cases in this work, it will be around 20,000.

The inputs and outputs of the substitution DNN model are those of the simulator. The continuous inputs are first normalized in $[0, 1]$ and directly fed into the DNN; The discrete inputs are one-hot encoded, i.e. transformed into one Boolean variable per possible value

of the discrete variable. The loss function is the Mean Squared Error (MSE) between the outputs of the DNN and the actual outputs of the simulator for the scenario at hand. Another quantity of interest is the accuracy of the substitution model, i.e., the proportion of scenarios correctly classified as G/NG. However, learning a Boolean classifier wouldn't have allowed to run an optimization algorithm to identify failures (see Section 4.5.3), giving little information to compare two different scenarios with same G/NG quality.

The architecture of the fully connected DNN is defined here by the number of layers, and the number of neurons per layer. It is tweaked based on the results on a validation set, mixing random search (Bergstra and Bengio, 2012) and grid search around the best randomly identified settings. The performance on the test set allows to monitor the goodness of the approximation, and as discussed, the accuracy in the G/NG classification is also reported.

4.4 The Reduced Models

As discussed, the tasks at hand can be set as optimization problems, that require to run the simulator within the optimization loop: the optimization algorithms need access to the outputs of the simulator for the candidate scenarios in order to compute the objective function being optimized. However, in the long run at least, this computation will require launching the simulator (SCANeR), which is computationally very costly: The simulation runtime of one scenario for the very simple use case described in Section 4.5.1 takes about 5 minutes. Hence, even if in the context of this work, we have replaced SCANeR with a substitution model that is very fast to compute (see next Section), we will act here as if we had to run the real SCANeR in the optimization loop.

We are hence facing a typical case of expensive optimization: we have to run an optimization loop while using the real simulator with parsimony. Classically, in the realm of Evolutionary Computation (Jin, 2011), we will build and use a surrogate model during the course of the algorithm, thus avoiding a systematic use of the actual simulation software. This model will output an approximation of the optimization criteria, and be updated during the iterations of the algorithm with the results of few actual simulations: only the scenarios that have been selected using the surrogate approximation will be checked with the real simulator. This surrogate model, built on the fly during the optimization, will be called, by analogy with current practice in numerical simulations, the *reduced model*.

4.4.1 Random Forests as Reduced Models

Because the reduced model will need to be re-built over and over again in the optimization, we need to choose a model that is fast to learn: while all possible choices of regression models are fast at inference time, Random Forests are amongst the fastest at training time, while being robust and reasonably accurate. Furthermore, they scale well enough with respect to the number of attributes (even though in this work only use cases with few attributes are addressed). We hence chose to use Random Forests for the reduced model.

After a few initial experiments, we decided to stick to the default values for the hyperparameters as follows: 100 trees of maximal depth 30. Because we had few attributes, it was not useful to select only some of them for the different trees. Throughout this work, we used Scikit-Learn (Pedregosa et al., 2011), the most popular Python Machine Learning library.

4.4.2 Initialization of the Reduced Models

First, Sobol sequences, which are low discrepancy sequences, are generated within the input ranges of the scenario parameters, being able to fill the space of scenarios more evenly than pseudo-random choice (Santiago et al., 2012). Then, some scenarios drawn uniformly from this set, are simulated to collect their criteria, and are included in the initial training set for the reduced model building, until a certain number of NG scenarios are retrieved: indeed, as we progress in the validation of the command law, most scenarios will be G, and we need examples of both classes in order to build the initial reduced model.

Most algorithms in the remaining of this thesis need some initial NG scenarios in order to bootstrap their search. In all cases, the required number of NG scenarios is set by the user, and the smallest possible initial set is built that satisfies this constraint.

Once such an initial set has been retrieved, the first instance of a reduced model can be learned. The reduced model is then used intensively by the algorithm in lieu of the actual simulator. At every iteration, some scenarios are selected depending on the task at hand (see Section 4.5 and 5.2). The real simulator (or the substitution model, see next Section) is then used on those selected examples, and their status (G or NG) is updated. They are then added to the training set with their real status, and, before starting next iteration, the reduced model is built again from this augmented training set.

4.5 Simulation-based Failure Detection

In order to illustrate the framework described in previous Sections, we will now present our first optimization algorithm **Find All Failures** dedicated to the direct discovery, in a very simple use case, of failures of the command law , i.e., input scenario parameters that lead to unsafe behavior of the autonomous vehicle. This is the first objective of this thesis, but the main objective remains to identify the borders of the failure region rather than isolated failure cases (see Chapters 5 and 6).

We will first introduce the use case, then instantiate the substitution model, before introducing the full optimization loop and presenting our results.

4.5.1 The Tracking Vehicle Use Case

Our first example of a use case is the simple *tracking vehicle* use case. The EGO car is following another vehicle in front of it, both are staying on the same lane, as illustrated in Figure 4.1. The preceding vehicle is scheduled to perform acceleration and deceleration

cycles during the simulation. The tracking vehicle use case has a dimension of 7, and all seven input variables of this use case are described in Table 4.1.

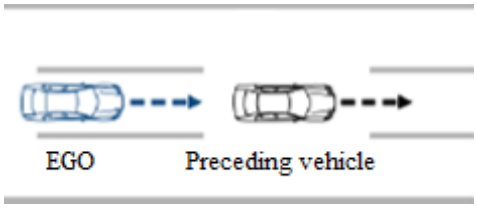

Figure 4.1: Tracking vehicle use case where EGO is tracking a preceding vehicle that is performing acceleration and deceleration cycles on the same lane.

Table 4.1: Description of the input parameters of the tracking vehicle use case.

Input parameters specifications		
Name	**Unit**	**Range/List**
Initial velocity of PV*	km/h	$[60, 110]$
Acceleration cycle value of PV*	m/s^2	$[1, 3]$
Deceleration cycle value of PV*	m/s^2	$[-3, 0]$
Acceleration/deceleration cycles time	s	$[0, 5]$
Initial inter-vehicle distance	m	$[3.5, 200]$
Type of PV*	-	Car, motorcycle, bus
Type of ground network	-	8 types of roads

*PV refers to the Preceding Vehicle in the use case.

During the simulation, the preceding vehicle performs acceleration and deceleration cycles as described, and the autonomous vehicle response is monitored throughout the scenario using the simulator outputs. For this tracking vehicle use case, three output variables are taken into consideration as detailed below. Each of them fits respectively into one of three test categories linked to the EGO vehicle: EGO position with respect to its environment, EGO dynamics, and EGO position with respect to other vehicles.

1. **Lateral lane decentering distance**: it aims at qualifying the lateral position of EGO in its lane, and calculates how far EGO is from the center of its lane. If the maximum decentering distance value reached throughout the simulation is higher (respectively lower) than a fixed threshold (e.g., 0.4015 meters for a road measuring 3.5 meters), then the criterion is labeled NG, (resp. G).

2. **Longitudinal deceleration**: this warning qualifies the stress suffered by a passenger due to a longitudinal deceleration in a straight line, and is defined as the worst perception among deceleration and jerk effects computed from lookup tables. It ultimately outputs an integer value that references the amount of passenger stress; 1, 2, 3 and 4 indicate comfort, dynamic, sport and emergency respectively. Because the goal is that

the passenger always feels comfortable while driving EGO, this criterion returns *NG* if the output value is higher or equal than 2, and *G* otherwise (i.e., equal to 1).

3. **Safety time gap**: this criterion monitors the distance between EGO and the preceding vehicle. In France, the threshold of the safety time gap is set by the Highway Code to 2 seconds (Breyer, 2010), below which the behavior of the car is considered to be unsafe. Therefore, this criterion detects a failure and returns *NG* if the time gap between EGO and the preceding vehicle is lower or equal than 2 seconds, and *G* otherwise.

As discussed, regardless of all other criteria, a single *NG* criterion qualifies the whole scenario as *NG*, but we will always look at the different criteria separately.

4.5.2 Building the Substitution Model

The Design of Experiment (see Section 4.3.2) consists of a total of 19,992 scenarios scattered in the input space parameters, and represents as effectively as possible various areas of input combinations that can be injected at the start of the simulations. These scenarios are input to the simulation software and their corresponding outputs are retrieved, building the training set for the substitution model.

After tweaking the DNN architecture as described in Section 4.3.2, we ended up with a NN architecture composed of 4 layers of 300, 200, 100 and 3 neurons respectively. The 3 neurons in the output layer correspond to the 3 warnings in the use case defined in Section 4.5.1. The loss chosen is the mean-squared error suitable for a regression problem. As for the other hyperparameters, the batch size chosen is 128, the learning rate is set to 0.005, and a Xavier initialization is chosen. Random search was used at first to detect optimal combinations of hyperparameters with good accuracies and a stable loss, then a finer grid search is conducted to identify the hyperparameters that maximize the accuracies obtained. Table 4.2 shows the accuracies obtained for the respective data sets for some neural network architectures and hyperparameters tested. For example, for a Xavier initialization, the global accuracies obtained are 98.61%, 96.13% and 96.01% for the training set, the validation set (used to choose the hyper-parameters, namely here the architecture of the network) and the test set respectively, whereas for a He initialization, we obtain 98.34%, 96.43% and 95.76% respectively. Since the global accuracies obtained with Xavier and He initializations are approximately similar, we choose the Xavier initialization where the accuracies are slightly better. Note that we were able to obtain acceptable test results without the need for regularization.

Furthermore, Figure 4.2 shows the evolution of the mean-squared error (chosen as cost function for this neural network) during the epochs for different architectures of neural networks. We can see that the cost function is quickly stabilized in all cases, with some minor fluctuations occurring for the bigger networks. Due to this stability, the number of epochs is fixed to 1000. Thus, since the biggest network achieved the best test accuracy with a good loss convergence, it is chosen as the surrogate model for this thesis.

Table 4.2: Accuracies of different neural network architectures and hyperparameters (the first line being the one chosen for this thesis).

Neural Network Architecture	Batch Size	Learning Rate	Initialization Method	Accuracies (in %)		
				Training	Validation	Test
300/200/100	128	0.005	Xavier	98.61	96.13	96.01
200/100/50	128	0.005	Xavier	97.26	95.15	95.41
100/50/25	128	0.005	Xavier	97.33	95.77	95.85
50/25/10	128	0.005	Xavier	96.78	95.61	95.24
300/200/100	64	0.005	Xavier	95.93	94.75	94
300/200/100	256	0.005	Xavier	98.52	96.06	95.61
300/200/100	128	0.001	Xavier	96.75	95.66	95.43
300/200/100	128	0.01	Xavier	99.16	95.29	95.06
300/200/100	128	0.005	He	98.34	96.43	95.76

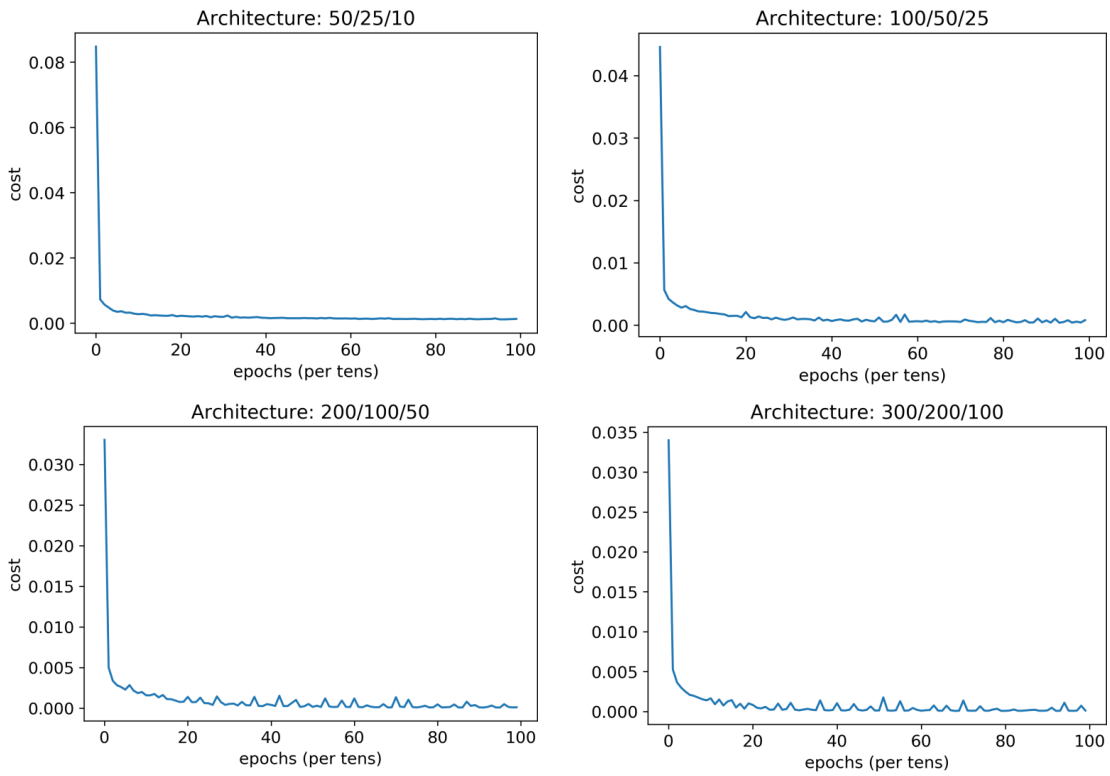

Figure 4.2: Evolution of the mean-squared error cost during the epochs for different neural network architectures tested.

Finally, we notice that 18 million parameters were needed in order to reach such acceptable accuracies, which reflects the difficulty of this problem. As discussed in the following of this work, this neural network will be used in lieu of the simulation software whenever a simulation is needed or mentioned, thus overcoming the current software limitations.

4.5.3 The Find All Failures Algorithm

The **Find All Failures** algorithm proceeds by repeatedly running an instance of a **Find One Failure** optimization algorithm, whose goal is to to identify one NG scenario that lies as far as possible from the known NG scenarios stored in the dynamic archive.

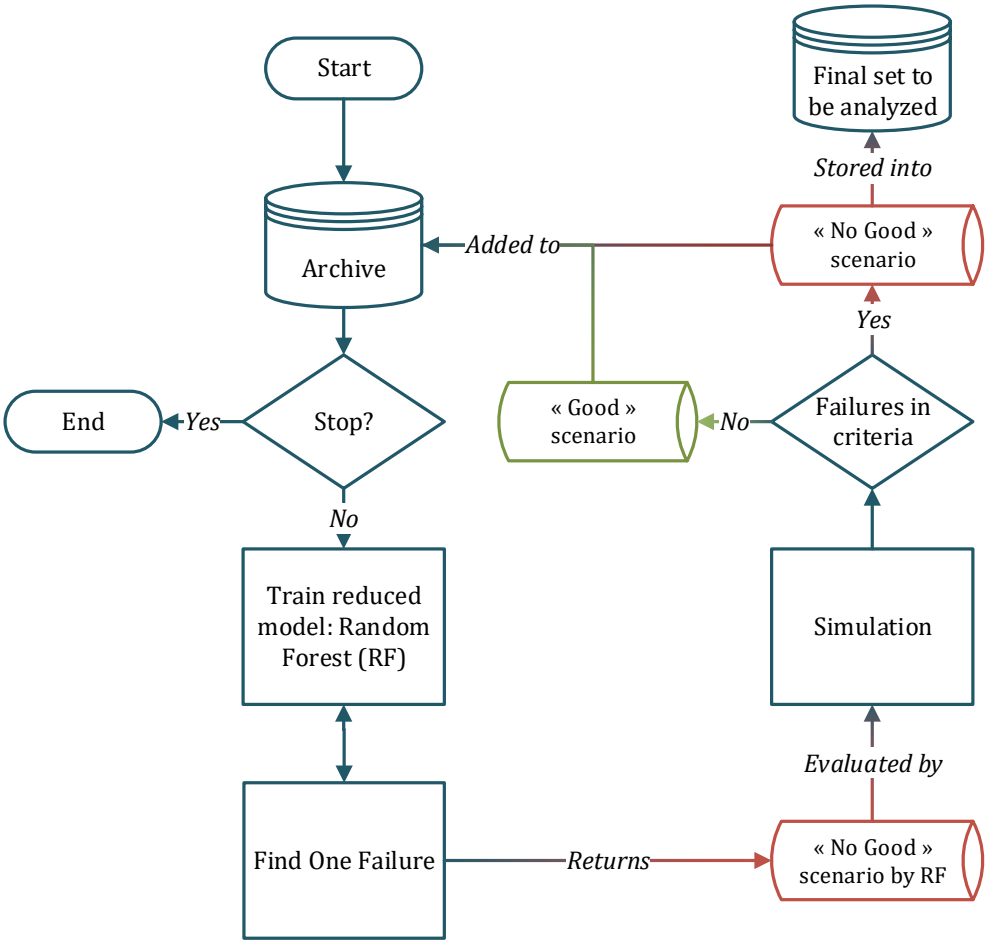

Figure 4.3: Flowchart of **Find All Failures**, which repeatedly detects new faulty scenarios lying as far as possible from the ones from the archive using the embedded **Find One Failure** optimization algorithm.

The flowchart in Figure 4.3 illustrates the structure of **Find All Failures**. First, the

initial set is built (Section 4.4.2) so as to it contains 30 NG scenarios and around 50 scenarios in total depending on the seed chosen, and a first Random Forest reduced model is trained (Section 4.4.1). The optimization algorithm **Find One Failure** is then launched, using intensively the Reduced Model to identify one NG scenario (according to the current reduced model). This scenario is then checked with the actual simulation (here, the substitution model, see Section 4.3), and is stored into the archive with its true criteria. If it is actually NG, it is also stored into another archive that will be the result of the algorithm after its stopping condition is met, and will be thoroughly analyzed (Section 4.5.5).

Obviously, during the first iterations, the accuracy of the reduced model will be limited, as it is built on very few examples. It is therefore important to check its decision G/NG with the ground truth. As iterations proceed, the reduced model will become more and more precise. This is further illustrated in Figure 4.4 which shows the number of prediction errors realized by the Random Forest model during the simulations, i.e., the number of scenarios that were predicted as NG by the Random Forest but turn out to be G when checked by simulation.

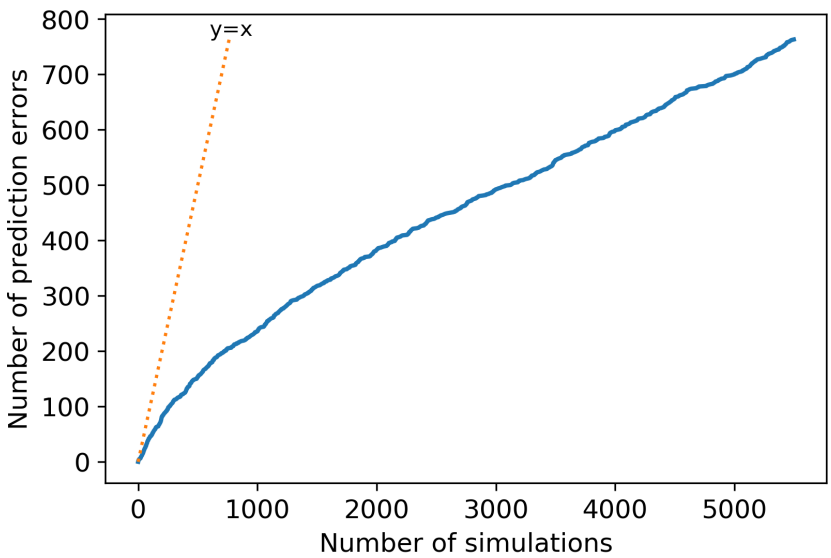

Figure 4.4: Evolution of the number of prediction errors related to the Random Forest model during the simulations. The stopping condition is reaching the minimal value of 0.15 a hundred times.

To understand this graph, we can compare it to the identity function $y = x$. In fact, each time the Random Forest model makes a wrong prediction, we count it as one occurrence of prediction error. Thus, the line $y = x$ represents the case where all Random Forest predictions were wrong when checked by simulation. At the first few iterations, we can see that the error trend is very similar to that of the identity function, meaning that the number of wrong predictions is significant during the first simulations launched. Then, as the iterations go by, the overall curve trend evolves in a sub-linear form and distances itself

from the identity function. Therefore, the Random Forest model is able to correctly predict *NG* scenarios throughout the iterations, gaining in accuracy each time scenarios are being verified by simulation.

Stopping condition The stopping condition of **Find All Failures** algorithm is based on the distance of the scenario returned by **Find One Failure** algorithm with the *NG* scenarios in the archive (i.e., the objective function of **Find One Failure** d_{obj}, see below): as the scenario space gets populated with *NG* scenarios, and even though **Find One Failure** tries to maximize it, the minimum distance of new *NG* candidates with archived *NG* scenarios globally decreases along the iterations. When it goes below some prescribed value for some prescribed number of time (in order to account for the numerical oscillations, see Section 4.5.5), the stopping condition is triggered. Several stopping conditions have been tried during this study and are presented, along with post-analysis and evaluation of the solution, in the next subsection.

A detailed description of **Find All Failures** can now be given in Algorithm 2.

4.5.4 The Find One Failure Algorithm

The goal of the **Find One Failure** algorithm is to detect one *NG* scenario as far as possible from the known *NG* scenarios in the archive, using the reduced model to assess the status *G*/*NG* of the scenarios it will consider.

Objective function In order to detect scenarios that lie as far as possible from the already known *NG* scenarios from the archive, we need to include in the objective function of the **Find One Failure** algorithm some cumulated measure of the distances between the considered scenario and all those in the archive, and to try maximizing this distance while making sure that the scenario is *NG*. Hence, a function that calculates this distance constrained by the *NG* condition is developed. It takes a scenario as input, and starts by predicting its criteria using the reduced model. The biased distance is calculated for each criterion using the following equations, where n is the dimension of the use case (and of the scenario space).

$$d_{NG}(x,y) = \begin{cases} \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}, & \text{if criterion is } NG \quad (4.1) \\ 0, & \text{if criterion is } G \quad (4.2) \end{cases}$$

If a criterion returns *NG*, this biased distance d_{NG} is simply the Euclidean distance between the normalized input scenario x and some scenario y of the archive that is *NG* for that same criterion (Equation (4.1)). Otherwise, this distance equals zero (Equation (4.2)).

Hence, *G* scenarios are penalized and *NG* scenarios are favored, since the algorithm will be pushing to maximize distance d_{NG}. The objective function $d_{obj}(x)$, to be maximized, is the minimum value of $d_{NG}(x,y)$, for all *NG* scenarios y in the archive.

Optimization engine The optimization algorithm used here is CMA-ES, which stands for Covariance Matrix Adaptation Evolution Strategy, the state-of-the-art of derivative-free continuous global optimization algorithms (Hansen and Ostermeier, 2001). Beside an objective function (the d_{obj} function described above), it requires an initial guess, and various parameters to be tuned e.g., the input bounds, the initial step size and some tolerance (its stopping criterion). The initial guess is drawn uniformly outside the archive, and its criteria are predicted by the reduced model. If a starting scenario is G through all its criteria, the distance computed will be zero, and the algorithm will choose another starting point. That is why NG scenarios are needed in the archive as mentioned in Section 4.4.2. Subsequently, the CMA-ES algorithm will perform multiple iterations maximizing d_{obj}, and returns the best candidate scenario with the value. More precisely, it goes through a cycle of proposing a scenario, predicting its criteria using the reduced model, and calculating its distance d_{obj}. Because the distance of the best scenario is not zero, at least one of its criteria reported a failure and the scenario is NG according to the reduced model.

Algorithm 2: Find All Failures Algorithm

- Initialize the archive (Section 4.4.2)

while *NOT stopping condition* **do**

- Pick a random initial scenario x outside of the archive
- Compute the minimum non-zero distance d_{obj} between the inputs of x and the inputs of the NG archive scenarios y across all 3 criteria using (4.1) and (4.2)
- Launch CMA-ES from this starting point to maximise d_{obj} while having access to scenarios criteria using the reduced model
- Evaluate the ground truth criteria of the new scenario obtained x_{new} by simulation and add it to the archive
- Learn a new Random Forest reduced model using the updated archive as training set
- Compare the maximum distance d_{new} obtained to the stopping condition value

end

4.5.5 Results

In this study, the preceding vehicle type is fixed as "car", and the road type as "normal" road whose track width measures 3.5 meters. The values of all five remaining inputs range between their corresponding intervals listed in Table 4.1. Hence, the dimension of the optimization problem is reduced from 7 to 5. Furthermore, the initialization of the Random Forest reduced model (Section 4.4.2) required 30 NG scenarios in the initial training set.

Figure 4.5 shows the evolution of the objective function d_{obj} throughout the algorithm iterations that successfully returned a NG scenario later evaluated by simulation. The stopping condition was set to 0.15 with count number 100, i.e., the algorithm stopped when the

best d_{obj} was below 0.15 at least 100 times.

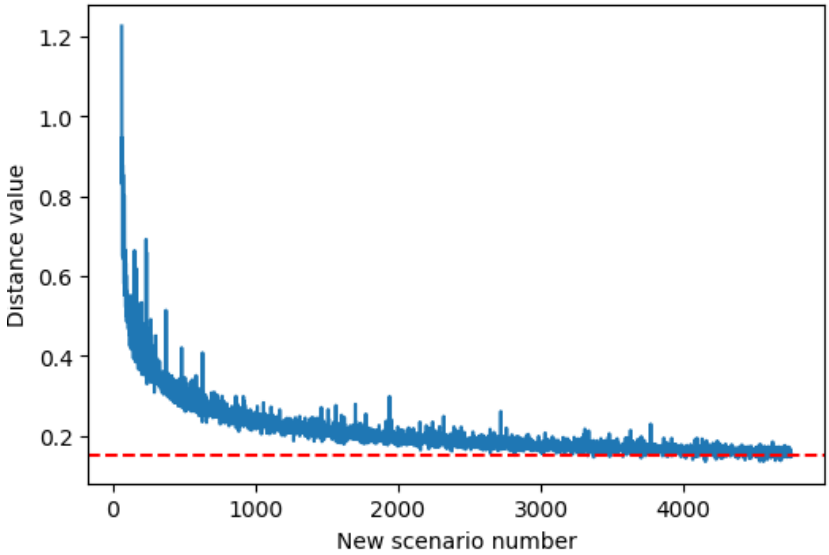

Figure 4.5: Evolution of the distance d_{obj} during the iterations of **Find All Failures** algorithm that actually resulted in a faulty scenario after evaluation by simulation. The stopping condition is reaching the minimal value of 0.15 a hundred times.

Numerical oscillation We notice in Figure 4.5 that the decrease of the objective function is not monotonous, and presents several oscillations. These are high at the beginning of the evolution, and start to decrease along the iterations. This can be explained due to the fact that the Random Forest reduced model is trained from only a few tens of scenarios during the first iterations, which means that its accuracy is initially poor. Plus, for each *NG* criterion, the distance function only considers the scenarios from the archive that have the same *NG* evaluation for that same criterion, which makes it highly dependent of the initial scenarios considered and their respective predictions as approximated by the Random Forest reduced model. Nonetheless, as the iterations go by, the reduced model is fitted into an ever-expanding set of scenarios with correct predictions by simulation. Therefore, it is updated continuously and its accuracy hopefully increases along the iterations. That is why the numerical oscillations tend to decrease throughout the process.

Several values of the stopping condition have been tried out, corresponding to precision values of 0.3, 0.25, 0.2 and 0.15. Furthermore, in order to account for the numerical oscillations, the stopping condition was triggered only after the threshold value has been overpassed several times, e.g. 10, 50, 100, 250 or 500, to make sure that the stopping condition was stably reached.

Evaluating the failure discovery rate In order to evaluate the quality of the resulting set of scenarios, it is compared to the *NG* scenarios of a full grid over the search space. Each

normalized input is discretized into 10 values, resulting in a total of 100,000 scenarios. These scenarios are actually simulated wth SCANeR, and their criteria are retrieved. The total number of *NG* scenarios found in the grid is 46,722 scenarios. Next, the idea is to take each *NG* scenario from the grid, and to calculate its distance with all the *NG* scenarios of the final set. If at least one of these distances is less than some fixed precision, it means that the algorithm was able to predict a *NG* scenario that is close to that grid scenario at this precision. The number of such instances can be viewed as a *discovery rate* up to the given precision. If the algorithm manages to predict scenarios close to all the grid scenarios within the precision distance, it has succeeded in exploring the whole search space and detecting all the failures (at the given precision).

To visualize the evolution of the discovery rates according to the number of simulations performed during a test, checkpoints were implemented throughout the iterations of **Find All Failures**. Each checkpoint takes place after a fixed number of simulations, and reports the corresponding proportion of the final set obtained. When the algorithm ends, discovery rates are calculated for each checkpoint according to the final set. These rates are then plotted against the number of simulations at each checkpoint. Furthermore, in order to effectively validate this stochastic algorithm, 11 independent runs are conducted for each stopping condition (varying the random seed, and hence the initial training set of the reduced model).

The failure discovery rates of the 11 final sets are computed, as well as their mean and standard deviations. The evolution of the mean discovery rate w.r.t the number of simulations is plotted on Figure 4.6 together with error bars (twice the standard deviation).

To compare the various stopping conditions, each test example stems from a different precision value of 0.3, 0.25, 0.2, and 0.15. Plus, we consider that the threshold distance required to calculate the discovery rate when compared to the grid should be equal to the corresponding precision of the stopping condition for each test example in this study. As a matter of fact, the distance value returned by the function represents how far away from the scenarios archive the new scenario proposed by the algorithm stands. However, since this distance is decreasing along the iterations until reaching a given precision, then the algorithm is managing to explore the search space only with respect to this prescribed precision value. So if we are assimilating the search space as a full grid, then we should calculate the distance between the *NG* scenarios introduced by the algorithm and the *NG* grid scenarios, and evaluate whether it is less than that **same precision distance** used by the algorithm as stopping condition. It is in that way that the discovery rate of the algorithm towards the grid is assessed.

Hence, we can see in Figure 4.6 that the mean discovery rate is increasing quickly during the first iterations, and continues to increase more slowly but steadily as the iterations go by, until reaching values almost equal to 90% and higher for all stopping criteria; it means that the algorithm succeeds in exploring the input search space. We also notice that the standard deviation values are the highest during the quick increase, corresponding to maximum values

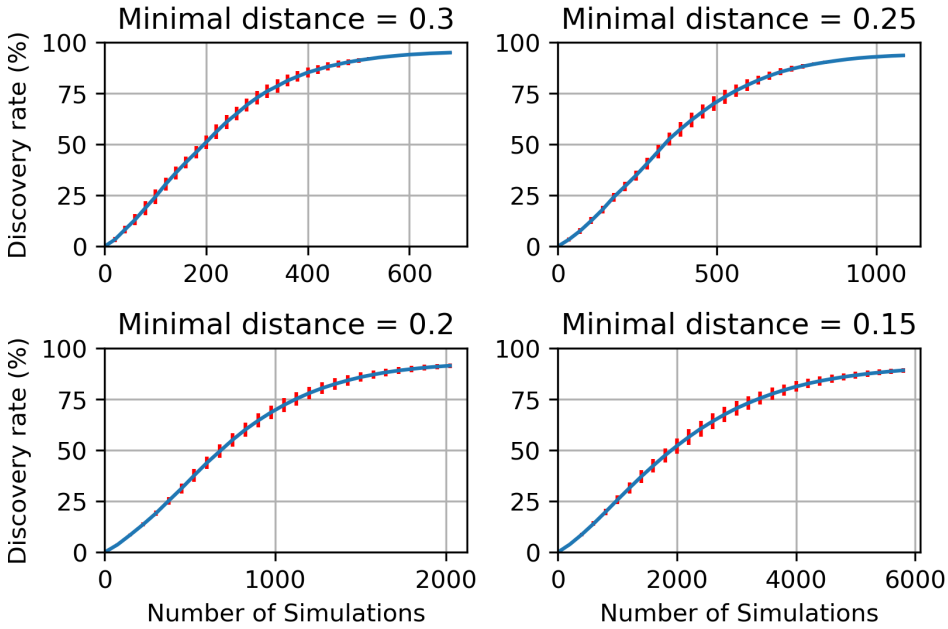

Figure 4.6: Mean discovery rates with std. dev. error bars w.r.t. the number of simulations. Each curve is the result of 11 independent runs. Four stopping conditions are represented: 0.3 (top left), 0.25 (top right), 0.2 (bottom left) and 0.15 (bottom right).

of less than 4% for all test cases, then begin to decrease throughout the iterations due to the reduced model that is gaining in accuracy.

The main idea to reflect upon, however, is that, because of the slower increase at high discovery rates, the highest the target discovery rates, the more we need simulations. Plus, if we compare the number of simulations between all test cases, we notice that the lower the precision of the stopping condition, the more simulations are needed in order to reach the same mean discovery rate. For instance, if we look at the algorithm performances in Table 4.3, 721 simulations are required in average for a precision of 0.3 to reach discovery rates almost equal to 90% and higher, against 1,161, 2,180 and 6,186 for precisions of 0.25, 0.2 and 0.15 respectively. Thus, because the key restriction of our study is to minimize the use of software simulations, a compromise should be made between attaining the best discovery rates, refining the search of failures, and calling the simulation software as little as possible for the aim of finding the maximum amount of failures.

Partial conclusion This section has introduced the first algorithm developed during this thesis. It improves the coverage of the simulated faulty scenarios and takes part in a series of algorithms intended to validate the command law of an autonomous vehicle. Its aim is to detect the maximum number of failures of its system during realistic virtual simulations. It has been experimentally validated on a tracking vehicle use case. Three criteria have been taken into consideration to determine whether there is a failure or not: the lateral lane

Table 4.3: Average algorithm performances on 11 runs for each example of stopping condition.

Stopping condition minimal distance	**Average algorithm performances**				
	Number of iterations	**Number of simulations**	**Time**	**Mean discovery rate**	**Maximum standard deviation**
0.3	1,968	721	2h49	94.99%	3.58%
0.25	3,113	1,161	5h37	93.62%	3.67%
0.2	5,835	2,180	8h41	91.48%	3.43%
0.15	16,621	6,186	33h23	89.22%	3.89%

decentering distance, the longitudinal deceleration and the safety time gap. The proposed algorithm has been designed to explore the input search space of the use case scenarios in order to detect the faulty ones. It iterates until reaching its stopping condition, which corresponds to a given precision for the distance. Because the search space is bounded and continuous, the success of the algorithm is measured by how well it managed to scan the search space effectively: this is achieved by comparing it to scenarios scattered on a full grid.

The results show that the algorithm manages to attain high discovery rates. For instance, 6,186 simulations are required in average to reach a discovery rate of almost 90% with a precision of 0.15. The scenarios obtained are evaluated whether they are close within a maximal distance of 0.15 from each scenario of a full grid of 100,000 scenarios. Thus, the **Find All Failures** algorithm is able to reduce the number of simulations needed to explore the input search space and detect directly faulty scenarios by comparison to the full grid which requires 100,000 simulations. Nonetheless, a compromise arises between reaching very high rates, which translates into detecting most failures accurately, and the constraint to launch as few software simulations as possible. It is worth noticing that, at the time this work was achieved, SCANeR calculations were very limited. As a consequence, this study (and the ADValue project) could only be based on this first use case. In particular, we were unable to test our approach on use cases of higher complexity. Nevertheless, all algorithms and models presented in this thesis are sent to the ADValue project, where they will be intensively used on future and more sophisticated use cases.

In the next Chapter, we will introduce three other algorithms that have been developed to tackle the second objective of this thesis: detect the border of the failure region. Since the inputs characterizing a use case can be continuous, we can visualize the input space as being divided into faulty and non-faulty zones rather than just being composed of scattered punctual scenarios. The aim of these algorithms will then be to detect scenarios that are situated near this border in order to locate it more easily. In fact, as the validation project advances, it is reasonable to assume that the number of *NG* scenarios, i.e. the number of failures generated by the autonomous system, should decrease until there is none at the project conclusion, marking the end of the validation for a given use case. Therefore, this first algorithm could not be enough to cover all these stages effectively, especially near the

project end, as it only operates on *NG* scenarios all the time, which then will become rarer to identify. This is where the idea of the ADValue project comes to light, which is to develop multiple algorithms that aid one another in the validation process through all the different project stages. If one algorithm fails to deliver efficiently at the next stage for example, other algorithms will try to overcome its limitations to guarantee a complete validation process. After all, the **Find One Failure** can also become an initialization process to the remaining algorithms, meaning it can detect certain *NG* scenarios initially for the other algorithms to function well.

Plus, a crucial goal of this thesis is to minimize the number of actual simulations needed, which is more efficient for industrial project monitoring purposes. Thus, a reduced model based on a Random Forest is fitted into a small initial set and used by **Find All Failures** as a proxy to detect faulty scenarios during the optimization. The simulation software (here represented by the substitution model), only evaluates a posteriori the optimal scenarios for the distance objective that have been predicted faulty by the reduced model. Both models will also be used within the optimization loop of the next three algorithms designed for border detection, which constitutes the second objective of this thesis.

Chapter 5

Border Detection

In this chapter, we will describe three algorithms that have been developed to help detecting the border between faulty and non-faulty zones in the use case input space, the second objective of this thesis. The rationale for this objective is the fact that some physical use case inputs, such as velocities and accelerations, are of continuous nature rather than a discrete list of possible value entries. The input space that characterizes the use case can then be seen as a partition into faulty and non-faulty areas, rather than just isolated scenarios in the input search space. Furthermore, in order to move from a faulty area to another non-faulty one, we will naturally have to cross the border that exists between both areas. Thus, if we can detect this border, then we can use it to directly identify new scenarios as G or NG by merely comparing their inputs to the border, after describing it more directly, the third objective of this thesis, to be addressed in next Chapter 6.

The first two algorithms, named respectively "**Find Border Max**" and "**Find Border Min**", approach the problem by seeking to identify pairs of G/NG scenarios which are close to one another in the input space: the border between a faulty and a non-faulty area should lie in between such pairs of close scenarios. The slight difference between both algorithms lies in the way they approach the border from an archive containing both G and NG points. The third algorithm, named "**Find Border Points**", is very similar to the **Find All Failures** algorithm presented in Section 4.5.3: It seeks to directly identify scenarios that lie "*on*" the border, or at least as close as possible to it, using an embedded optimization algorithm based on CMA-ES, **Find One Border Point**.

In any case, all three algorithms also have to follow the same industrial constraints than the algorithm for failures exploration of the previous chapter: using as little as possible the simulation software, replacing it with some reduced model updated on the fly. We will show their respective methodologies in detail before presenting the results obtained with each one of them, as well as the metrics used in order to evaluate and compare their performances.

Precision All three algorithms are required to respect a common minimal precision distance value in the search space. The distances between the new scenarios proposed, as well as

between these scenarios and all the archive of evaluated scenarios, should not exceed this precision value. This condition is defined in order to reduce as much as possible running simulations of scenarios that are too close, or even almost coincide, in the input parameters space.

One criterion at a time Because we have a multi-criteria use case, each criterion will define its own border depending on the G and NG areas for this particular criterion. Thus, the algorithms tackle each criterion one at a time, allowing us to evaluate the difference in detecting the border depending on the criterion considered. However, in order not to use precious actual computations in the real operational context, and because a single computation computes all three criteria anyway, the three criteria are processed in parallel, sharing one single archive used as training set for the reduced model. The results are of course stored separately.

Stopping Conditions Choosing a stopping condition for this study turned out to be more challenging than for the NG detection algorithm. In fact, the aim is to define a stopping condition that is shared between all three border detection algorithms in order to be able to effectively compare their performances. However, their objectives in detecting border scenarios are very different (detecting on-the-border scenarios vs identifying G/NG pairs), which complicates the task of setting one shared stopping condition. Therefore, we finally decided that the number of simulations (the number of neural network calls in this work) should be used to define the stopping condition: this is compatible with the industrial project constraints that aim at limiting the number of simulations in order to minimize the computing power used. This will allow a fair comparison of all results. In section 5.3 for instance, two series of experiments will be compared, using respectively 1,000 and 3,000 simulations.

We will now introduce two types of algorithms for border detection. The first type handles pairs of scenarios, trying to identify G/NG pairs that are close enough to ensure that only one simple border between faulty and non-faulty regions is located between them. The second type directly targets scenarios that lie on the boundary, up to a given threshold on the criteria.

5.1 Find Border Pairs

This Section introduces two algorithms, named **Find Border Max** and **Find Border Min**, that both try to identify pairs of scenarios such as one is G and the other one is NG, and the distance between them is small enough so that we can assume that a simple border (variety of dimension $n-1$ if n is the dimension of the scenario space) between faulty and non-faulty regions lies between them. Both algorithms share the same global methodology, that resembles that of **Find All Failures**, with some differences due to the presence of pairs of scenarios as solutions.

5.1.1 Methodology

This flowchart of both algorithms is illustrated in Figure 5.1. First, the archive is initialized with an initial set similarly to the process described in Section 4.4.2. A fixed number of NG is defined, and we randomly draw scenarios and add them to the initial set until that number is achieved. In this chapter, we considered initialisation sets that follow this methodology with 50, 100 and 500 *NG* scenarios. With the current command law, the whole initial sets would amount to approximately twice the number of *NG* scenarios. As the command law improves, it will be gradually more and more difficult to find *NG* scenarios through random search. An alternative will be to use an optimization tool, like CMA-ES, to identify the prescribed number of *NG* scenarios - though at a cost of several additional evaluations. Furthermore, it will be necessary to discard some *G* scenarios to keep the initial set balanced.

A first Random Forest reduced model is then trained (Section 4.4.1) from this initial training set. Within this archive, *G* and *NG* scenarios are separated. They are then used to discover new *G*/*NG* pairs with a specific procedure for each algorithm, as will be described below. Both procedures make an intensive use of the Random Forest reduced model. This pair of *G*/*NG* scenarios (according to the reduced model) is first checked to see if it is further away than the precision distance d_{min} from existing solutions in the solution archive, in which case it is evaluated with the actual simulation (here, the substitution model, see Section 4.3), and is stored into the archive with its true criteria. If it is confirmed by simulation that it is actually a *G*/*NG* pair, it is also stored into another archive, the *result archive*, that will be the result of the algorithm after its stopping condition is met, and will be thoroughly analyzed (Section 5.3). Such iteration is done for all three criteria, sharing the same archive used to train the reduced model, as explained at the beginning of this Chapter.

We will now detail for both algorithms the way they identify new *G*/*NG* pairs from the archive. First of all, we partition the archive in two sets, one of *G* scenarios and one of *NG* scenarios, and we compute all distances between *G*/*NG* pairs.

5.1.2 The Find Border Max Algorithm

We first identify the *G*/*NG* pair of scenarios $(\mathbf{X_G}, \mathbf{X_{NG}})$ that are the farthest apart from each other in the scenario space (hence the name **Find Border Max**). These two scenarios are the initial starting points of the procedure. We then consider N_{sce} linearly spread scenarios on the $[\mathbf{X_G}, \mathbf{X_{NG}}]$ segment, i.e., defined by $N_{sce} = \frac{||X_G - X_{NG}||}{d_{min}}$, where d_{min} refers to the minimal precision distance value, using the following equation:

$$\mathbf{X_p} = \mathbf{X_G} + \frac{p+1}{N_{sce}+1}(\mathbf{X_{NG}} - \mathbf{X_G}), p \in [0, N_{sce} - 1] \tag{5.1}$$

Because $(\mathbf{X_G}, \mathbf{X_{NG}})$ is a *G*/*NG* pair, at least one of the $(\mathbf{X_p}, \mathbf{X_{p+1}})$ pairs is also a *G*/*NG* pair. We now consider all such pairs on the $[\mathbf{X_G}, \mathbf{X_{NG}}]$ segment as candidate pairs for the **Find Border Max** algorithm. However, remember that these scenarios have been qualified as *G* or *NG* using the reduced model. Because we know that the reduced model might be

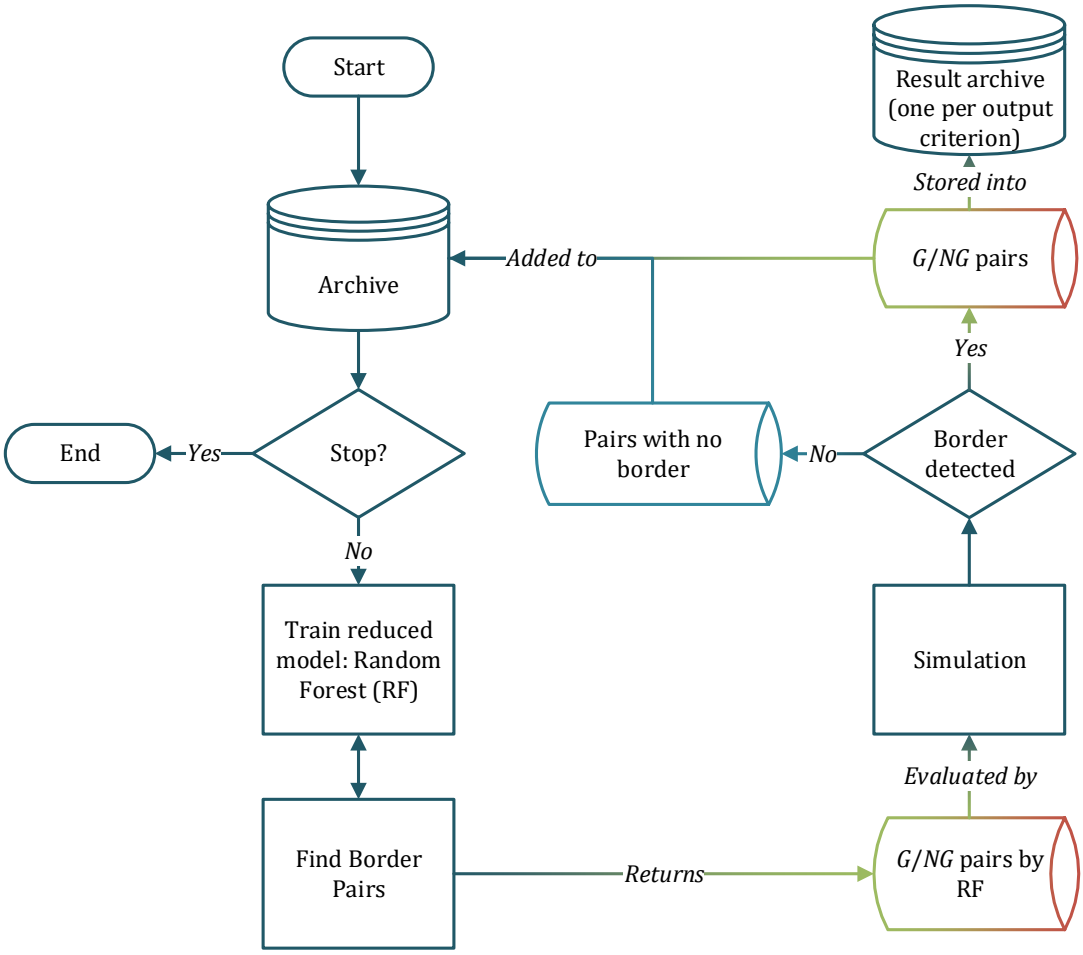

Figure 5.1: Flowchart describing the framework of the G/NG pairs detection algorithms **Find Border Max** and **Find Border Min** while using the simulation software as little as possible.

poorly accurate, especially during the first iterations of **Find Border Max**, when it was built using very small training sets, we need to check their ground truth status with the simulator (substitution model here). But first, the distance of all these G/NG pairs with all pairs from the result archive is computed, and the pairs that are closer than d_{min} from one saved pair are discarded. The remaining pairs are fed to the simulation (substitution model). All these new pairs are then added to the archive, and if they are confirmed as G/NG by the simulation, are added to the result archive for later analysis.

The importance of the initial set (first archive) is crucial here: the exploration of the scenario space by **Find Border Max** will only take place in the convex hull of this initial set. Several initial conditions have been tried during this study, and are presented in the

Section 5.3, where the results of all border detection algorithms will be compared.

Algorithm 3 summarizes the **Find Border Max** Algorithm.

Algorithm 3: Find Border Max

- Initialize the archive (Section 4.4.2)

while *NOT stopping condition* **do**

- Retrieve the maximally separated G/NG pair of scenarios from the archive
- Generate the scenarios on this G/NG segment according to Equation (5.1)
- Identify all G/NG pairs on the segment according to reduced model
- Discard those that are closer than d_{min} from a pair in the result archive
- Compute the real criteria of the other G/NG pairs, add them to the archive
- Store in the result archive all confirmed G/NG pairs, criterion per criterion
- Train the Random Forest model anew using the archive as training data

end

5.1.3 The Find Border Min Algorithm

From the same starting point than **Find Border Max**, the **Find Border Min** Algorithm proceeds differently to identify new G/NG candidate pairs from the archive.

We now retrieve from the archive the G/NG pair with the smallest distance between them (hence the name). We proceed by dichotomy from this initial pair, evaluating the status of the middle point (using the reduced model), and keeping the half-interval that has G/NG endpoints, until reaching the precision d_{min}: The dichotomy stops when the length of the interval is less than $2 * d_{min}$ (but still larger than d_{min}). When the dichotomy ends, the distance of the current G/NG pair with all pairs from the result archive is computed, and the pair is discarded if closer than d_{min} from one saved pair. Otherwise, it is input to the simulation (substitution model). The new pair is then added to the archive in that case, and added to the result archive for later analysis if it is confirmed as G/NG.

Like **Find Border Max**, **Find Border Min** only explores the convex hull of the initial set, and care must be taken when building it, as already suggested, and as will be demonstrated in Section 5.3.

Algorithm 4 summarizes the **Find Border Min** Algorithm.

5.2 Find Border Scenarios

5.2.1 The Find Border Points Algorithm

The general methodology of this third border detection algorithm is very similar to that of the **Find All Failures** algorithm introduced in Section 4.5.3: it proceeds by repeatedly

Algorithm 4: Find Border Min

- Initialize the archive (Section 4.4.2)

while *NOT stopping condition* **do**

- Retrieve the G/NG pair of scenarios with minimum distance between them
- Launch the dichotomy procedure until finding a G/NG pair with distance close to d_{min}
- Evaluate the real status of this last pair, and add it to the archive
- If confirmed G/NG pair, add it to the result archive, criterion per criterion
- Train anew a Random Forest model using the archive as training data

end

running an instance of a **Find One Border Point** optimization algorithm, whose goal is to identify one scenario that exactly lies **on** the boundary.

The general structure of **Find Border Points** is hence that of Figure 4.3 where **Find One Failure** is replaced by **Find One Border Point** as the embedded optimization algorithm (bottom left box). **Find One Border Point** returns a "border scenario" as evaluated by the reduced Random Forest model, while respecting the minimal precision distance d_{min} with the archive scenarios, and the generated scenario is evaluated by the simulator (i.e., the substitution model for the time being). If the real criteria confirm that this scenario actually lies on the border, it is stored in the final set to be analyzed, and in any case, it is added to the archive, that is the training set to update the reduced model at next iteration. The main difference with **Find All Failures** lies of course in the objective function of the embedded **Find One Border Point**, that we will detail now.

Identifying the Border As said, the goal is to detect scenarios which are on the border, and the border is defined by threshold values of all output criteria. The objective function should hence take into account the distance to these thresholds in the output space. The simplest possible objective function is hence, for a given criterion i:

$$gap_i(x) = \mid y_i - t_i \mid \tag{5.2}$$

where y_i is the i-th warning criterion of x (be it evaluated by the Random Forest reduced model of the substitution model), and t_i is the corresponding threshold for that criterion.

Therefore, the goal of **Find One Border Point** should be to minimize the gap between the criterion evaluation and its defined threshold, which should ultimately result, if the optimization is performing well, in scenarios whose criterion is very close to its threshold, i.e. scenarios that are on the border related to that specific criterion.

However, because we cannot wait until hitting exactly the boundary, especially due to the use of the reduced model, we need to extend the concept of boundary to a region of the scenario space with non-zero measure. We hence define a range around every threshold,

that we call *border boundary*, and consider that a scenario lies "on" the boundary for a given criterion if the gap with the corresponding threshold (Equation 5.2) falls within this range. The border boundaries have been fixed to a value of 10% around the threshold for the continuous output criteria. As for the discrete longitudinal deceleration variable, it is more challenging to define a border boundary, since this output criterion outputs G iff it is equal to 1, and NG in all other cases ([2,4]). Taking the whole interval $[1, 2]$ as the border boundary would result in all the G scenarios being considered as on the border. We hence decided to take into consideration the interval $[1.5, 2.5]$ as the border boundary instead, so as to only take the neural network predictions close to the threshold value equal to 2. The values of these border boundaries are resumed in Table 5.1.

Denoting the border boundaries of criterion i as $[lb_i, ub_i]$, the objective function, to be minimized (with minimum 0 if within the bounds of the border boundaries), becomes for criterion i:

$$obj_i = min\{(lb_i - y_i)^+, (y_i - ub_i)^+\} \tag{5.3}$$

It is worth noticing that, contrary to the **Find All Failures** algorithm, the objective function of the **Find One Border Point** optimization algorithm is computed w.r.t. the outputs of the simulation, and not its inputs (the scenario parameters). Hence the only coupling between successive runs of **Find One Border Point** within the main **Find Border Points** loop is achieved through the archive and, consequently, through the reduced model, that takes into account all identified points in its training set.

The Find One Border Point Algorithm Like **Find One Failure** (Section 4.5.4), **Find One Border Point** uses CMA-ES to optimize its objective function, i.e., for the chosen criterion, Equation (5.3) where criterion y_i is evaluated by the reduced model. As for **Find One Failure**, all parameters of CMA-ES should be carefully tuned. However, the most crucial one here is its starting point, i.e., an initial scenario randomly drawn outside the archive of evaluated scenarios.

Indeed, we want to use CMA-ES here to find multiple scenarios on the border, spread on the whole use case input space. In particular, we want to avoid always detecting the same scenarios which represent global optimums for the objective function, or at least those local optima with large basin of attraction. A possible approach could have been to penalize the objective function with the distance to the already discovered scenarios on the boundary, as was done when designing the objective function of **Find One Failure** (see Section 4.5.4). However, the optima of such penalized fitness function would have been trade-offs between both goals: minimize the gap with the border (Equation 5.2) and maximize the distance to previously discovered scenarios on the border. Another approach was chosen: use CMA-ES as a local optimization algorithm by setting its initial step size to a very small value = 0.01, and thus identify the scenario on the border which is the closest to the starting scenario, and spread these starting scenarios uniformly over the scenario space. This value was fixed after considering many values and evaluating the optimization process if it performs well

in detecting close border scenarios. Furthermore, if the optimal scenario returned by **Find One Failure** is too close to existing scenarios in the archive (less than d_{min}), it is simply discarded, and no simulation occurs during this iteration.

Another of CMA-ES parameters is the tolerance, that is used as a stopping criterion, and decides on the required precision of the objective function. However, every criterion has its own unique type and range, which should be taken into consideration if we want to improve the precision for the borders of all criteria. For instance, in the simple use case under study here (Section 4.5.1), the longitudinal deceleration is a discrete state defined by an integer between 1 and 4, whereas the safety time gap is a variable in seconds defined by a positive real number. Therefore, the tolerance is tuned w.r.t the corresponding criterion on which the objective function is defined for the optimization algorithm. For each criterion, we decided that the tolerance should allow a precision of approximately 10% of the gap between the threshold and the upper or lower bounds, to make sure that detecting a scenario within border boundaries resists to small changes in the optimization. For instance, for the safety time gap with border boundary of 0.2, it is equal to 0.02. However, we should also take into consideration that this value has to be also normalized according to the normalization process across the different outputs to obtain a fair model across criteria. Thus, the final tolerance values injected to CMA-ES for each criterion in this work are found in Table 5.1.

Furthermore, an optional parameter in CMA-ES triggers a number of restarts of the optimization process with the same initial scenario as starting point. After completing the requested number of restarts, CMA-ES outputs the scenario with the best results out of all restarts. This parameter increases the probability of the optimization to perform well and reach a border close to the initial scenario. It has been set to 2 in this work.

We can now give a more detailed overview of **Find Border Points**, in Algorithm 5.

Table 5.1: Output criteria border boundaries and tolerances for **Find Border Points**.

Output criterion	**Type**	**Threshold**	**Border Boundaries**	**Tolerance**
Lateral lane decentering distance	Continuous	0.4015	$[0.36, 0.44]$	0.0025
Longitudinal deceleration	Discrete	2	$[1.5, 2.5]$	0.01
Safety time gap	Continuous	2	$[1.8, 2.2]$	0.0025

5.3 Experimental Results

Goal of Experiments The goal here is to compare the three algorithms defined in previous Sections 5.1 and 5.2 along two main directions: the computational cost induced by using one or the other to identify scenarios on the border of the failure region, insofar that this is an imposed restriction from the project specifications, and, more importantly, the global quality of the scenarios that are identified by each algorithm, keeping in mind that our ultimate goal will be to precisely identify the border (see Chapter 6).

Algorithm 5: Find Border Points

- Initialize the archive (Section 4.4.2)

while *NOT stopping condition* **do**

- Pick a random initial scenario outside of the archive
- Run **Find One Border Point** algorithm starting from this initial scenario to minimize the gap using the Random Forest reduced model
- If the returned solution is at more than d_{min} from all points of the archive
  - Evaluate its true criteria using the simulation, and add it to the archive
  - If this new scenario really is on the border, add it to the result archive, criterion per criterion
- Learn a new Random Forest reduced model using the updated archive as training set

end

The main hyperparameter to be varied to tune the three algorithms is the initial set used to seed both border pair detection algorithms **Find Border Max** and **Find Border Min**, as its influence is expected to be crucial, defining the region where pairs can be identified (note that this initial set is not expected to have much influence on the results of the **Find Border Points** algorithm).

5.3.1 Experimental Conditions

The Use Case The same "Tracking vehicle" use case than in Section 4.5.1 was used here, and as in Section 4.1, the dimension of the optimization problem is reduced from 7 to 5: both input variables "preceding vehicle type" and "road type" are fixed to "car" and "normal road" respectively. The values of the five remaining inputs range between their corresponding intervals listed in Table 4.1. There are the same three output criteria linked to the EGO vehicle: the lateral lane decentering distance, the longitudinal deceleration state, and the safety time gap

The Stopping Condition Choosing a stopping condition for this study turned out to be more challenging than for the **Find All Failures** algorithm. In fact, the aim is to define a stopping condition that is shared between all three border detection algorithms, in order to be able to effectively compare their performances. However, their objectives for detecting border scenarios (identifying G/NG couples or *on*-the-border scenarios) differ, which complicates the task of setting one shared stopping condition. Therefore, we finally decided that the number of simulations (i.e., the number of calls to the substitution model in this manuscript) should be the only stopping condition: it also matches the industrial project objectives and constraints on the number of simulations allowed in order to minimize computing consumption. A first stopping condition of 1,000 simulations was imposed, which

is in par with the number of simulations that can be restricted by the industrial project, and a series of comparative experiments of the three algorithms was launched in this context and its results are detailed in Sections 5.3.2 and 5.3.3. Another series of experiments with a stopping condition of 3,000 simulations was investigated too, in order to study if the previous findings still hold for a much longer computing budget, and its results are briefly presented in Sections 5.3.4 and 5.3.5.

The initial sets First of all, it is clear that the results of **Find Border Points** in its current version depend only marginally on the initial set when it comes to evaluate the quality of the results, and this algorithm will be studied for a single initial set for the quantitative results (i.e. time needed and number of scenarios obtained).

Because both **Find Border Max** and **Find Border Min** algorithms look for pairs of G/NG scenarios, even if using different ways of exploring the scenario space from the initial set, this initial set should contain approximately the same number of G and NG scenarios. Furthermore, the complete process of the command law validation iterates over the steps described in this thesis while gradually modifying the command law to decrease its failure zone in the scenario space. It is hence hoped that less and less NG scenarios will exist as the process goes on – even though only the data for one single step of this overall process (i.e., one single instance of command law) was available at the time these experiments were made. It was hence decided to use the number of NG scenarios it contains as a kind of measure of complexity of the initial sets used in the experiments, knowing that enough G scenarios will always be easier to gather. Three such complexities have hence been launched, with initial distribution sets containing 50, 100 and 500 NG scenarios for each algorithm, with approximately the same number of G scenarios drawn according to the current version of the command law. Furthermore, because we know that both **Find Border Max** and **Find Border Min** algorithms only search in the convex hull of the initial set, and because the dimension of the use case allowed it here, a fourth initial set containing all corners of the scenario space was added to the experiments – while being impractical with realistic dimensions of the scenario space.

Finally, as all three algorithms are stochastic (and in particular for **Find Border Max** and **Find Border Min**, the choice of the initial set), for each condition on the initial set (i.e., number of NG scenarios, of corners), in order to test the statistical validity of our findings, all results report means and standard deviations (vertical bars around the means) over 11 independent runs, i.e., in which different random seeds of the pseudo-random number generator were used, including for the choice of the initial sets when it matters.

Performances Measures All experiments are made on the Titanic cluster maintained by INRIA, by using the SLURM batch scheduler, which is lightweight and efficient in dispatching jobs on the various tens of nodes of the cluster, with around 48 CPU threads and 12 GB of RAM each.

The performances measures include quantitative results, such as the time needed by each algorithm to accomplish 1,000 and 3,000 simulations, and the number of pairs or border scenarios generated, as well as qualitative results to assess if the scenarios obtained are able to effectively detect the border in the search space. Besides, the accuracy of the Random Forest reduced model has also been monitored, as in Section 4.5.3, witnessing the progress of the process.

5.3.2 Quantitative Results – 1 000 simulations

Computational Cost

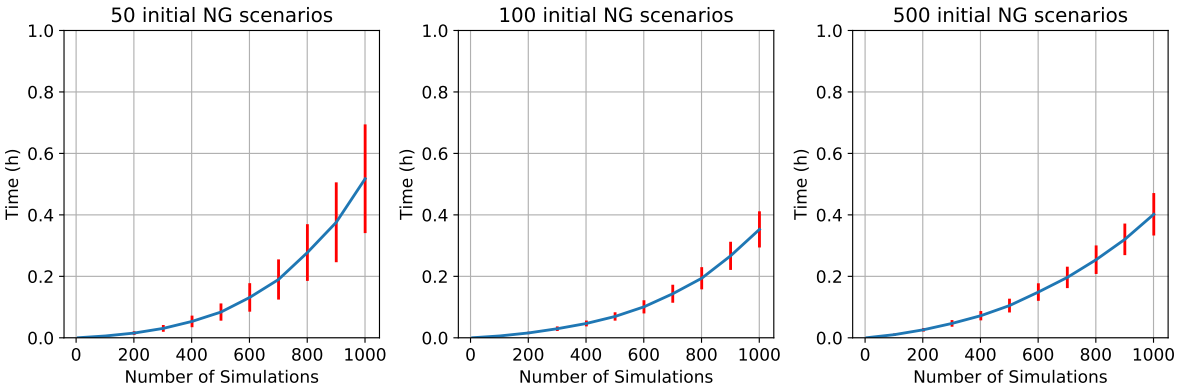

Figure 5.2: Average time (hours) and standard deviations (vertical bars) spent by **Find Border Max** to reach 1,000 simulations. From left to right, initial sets containing 50, 100 and 500 NG scenarios.

Figure 5.2 shows the time evolution (in hours) for the initial distributions sets containing 50, 100 and 500 NG scenarios. **Find Border Max** needs less than one hour to complete 1,000 simulations for all initial circumstances, despite demonstrating a super-linear behavior. We can also notice that the increase of number of scenarios from 50 to 100 in the initial set speeds up the completion of the algorithm, while no significant difference is observed when going from 100 to 500. Indeed, it introduces as many scenarios as possible between the widest G/NG pairs, all at once while respecting the minimal precision distance d_{min}: **Find Border Max** algorithm introduces new scenarios proportionally to the maximum G/NG distance encountered in the current archive. Therefore, the larger the initial G/NG distance at each iteration, the more chances of detecting a border by the reduced model, increasing in turn the number of scenarios to be simulated. Hence, this number of simulations per iterations increases with the size of the initial set, as this will provide G/NG pairs with higher distances, hence opening more opportunities for the algorithm to propose new scenarios.

This is confirmed by Figure 5.3, that similarly presents the time spent by **Find Border Max** to reach the 1000 simulations when starting from the initial set containing all 32 corners of the scenario space. This experiment is the fastest among all initial sets experienced with

to complete 1,000 simulations, as could be expected.

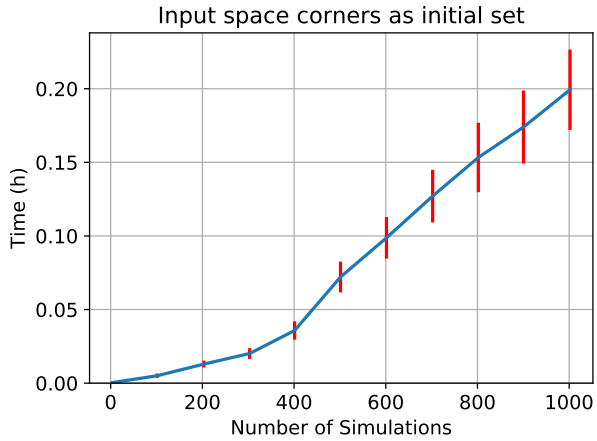

Figure 5.3: Average time (hours) and standard deviations (vertical bars) spent by **Find Border Max** to reach 1,000 simulations for the initial set made of all 32 corners of the scenario space.

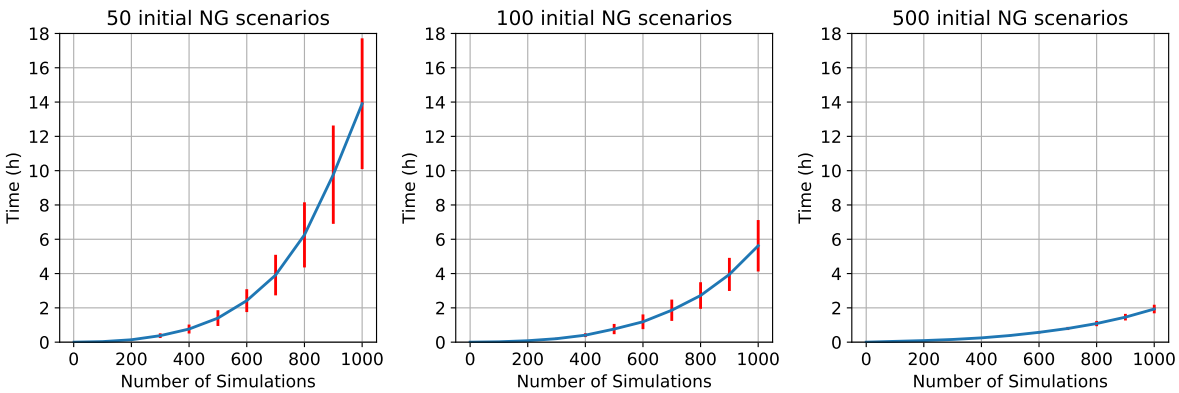

Figure 5.4: Average time (hours) and standard deviations (vertical bars) spent by **Find Border Min** to reach 1,000 simulations. From left to right, initial sets containing 50, 100 and 500 *NG* scenarios.

The behavior of **Find Border Min** algorithm is clearly different, as can be seen on Figure 5.4: While **Find Border Max** needs less than one hour in all cases, **Find Border Min** needs 14 hours in average with an initial set containing 50 initial *NG* scenarios, with rather large standard deviations, increasing a lot with the number of simulations. Furthermore, the time to 1000 simulations decreases the more we expand the initial set: an average of 6 and 2 hours respectively for initial sets containing 100 and 500 initial *NG* scenarios. All curves are also super-linear w.r.t. the number of simulations.

This difference in running time can be explained by the intrinsic difference between both algorithms: As opposed to **Find Border Max** that introduces many scenarios at each itera-

tion, **Find Border Min** proposes a single scenario for each output criterion at each iteration. Hence, it will need more iterations (and time) to achieve a fixed number of simulations than **Find Border Max**. Furthermore, if the single scenario proposed at each iteration does not respect the minimal precision distance d_{min}, the algorithm will re-iterate and find a new pair of scenarios, which will increase even more the number of iterations needed.

This is confirmed more clearly by looking at the overall number of iterations needed for both algorithms, as visualized in Figures 5.5 and 5.6.

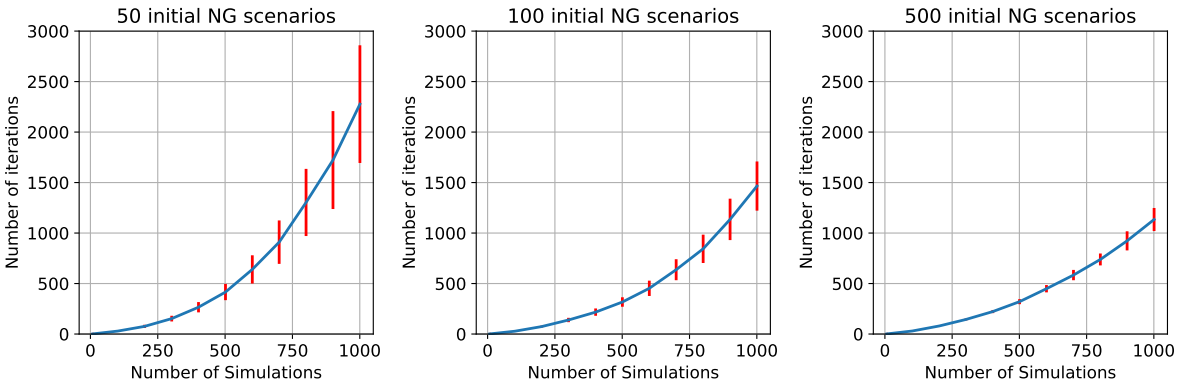

Figure 5.5: Average number of iterations (and standard deviations) needed by **Find Border Max** to reach 1,000 simulations. From left to right, 50, 100 and 500 *NG* scenarios.

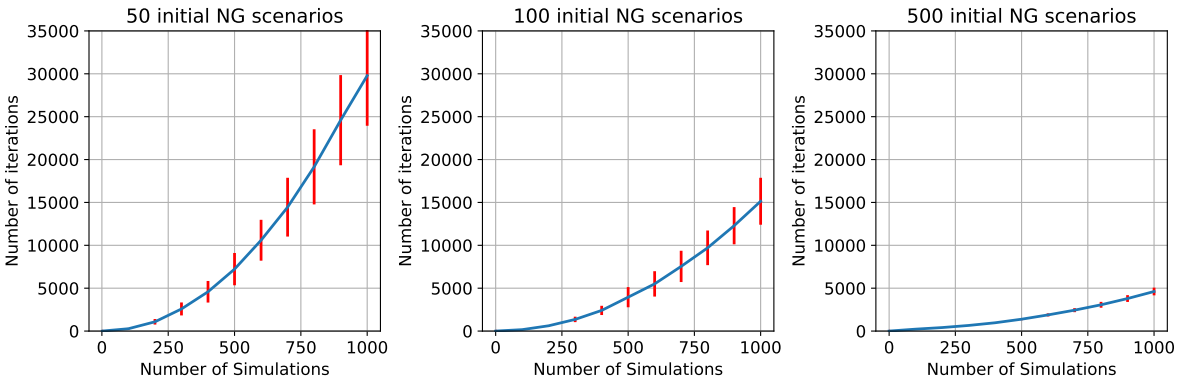

Figure 5.6: Average number of iterations (and standard deviations) needed by **Find Border Min** to reach 1,000 simulations. From left to right, 50, 100 and 500 *NG* scenarios.

These two figures also exhibit super-linear behavior, demonstrating the direct link between the number of iterations and the average computing time. Additionally, the number of iterations needed by the **Find Border Min** is almost ten times higher than the one for **Find Border Max** for all initial sets, which further reflects the disparity in choosing the maximum or the minimum distance between the G/NG scenarios and the consequences of this choice.

Moreover, we also ran **Find Border Min** with the corners initial set to see if it also

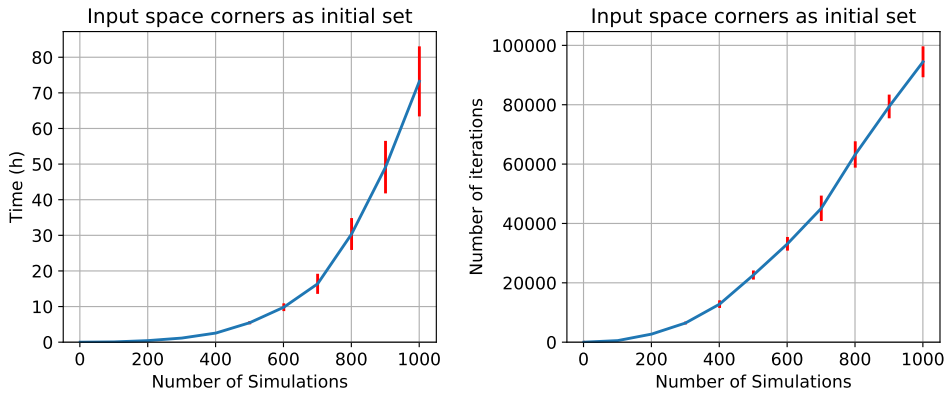

Figure 5.7: Average amount of time in hours (left) and number of iterations (right) spent by **Find Border Min** to reach 1,000 simulations with vertical error bars equal to twice the standard deviation values. Each curve is the result of 11 runs with the corners of the use case input space as initial set.

impacts its performance. Figure 5.7 shows both the average amount of time and number of iterations which were needed to accomplish a thousand simulations. Here, the algorithm performed nearly 100,000 iterations in more than 70 hours before reaching 1,000 simulations, while **Find Border Max** achieved that same goal in approximately 10 minutes! Therefore, the initial sets greatly impact the time performance of both algorithms, whether it contains uniform samples of G/NG scenarios or the corners of the input search space. Later on, we will analyse the quality of the results to better assess their value while taking into consideration the time they each had to consume to reach their common goal.

Finally, the third algorithm **Find Border Points** uses CMA-ES to directly propose scenarios "*on*" the border using the border boundaries defined for each output criterion as detailed in Table 5.1. Since it depends only marginally on the initial set, the quantitative results are only shown for the initial set containing 100 NG scenarios. Figure 5.8 shows the time consumed by the algorithm to attain 1,000 simulations.

We can see in Figure 5.8 that the time evolution curve is rather linear this time instead of being super-linear like the curves illustrated by the **Find Border Max** and **Find Border Min** algorithms, taking approximately 12 hours to complete 1,000 simulations. This linear aspect of this curve is directly linked to the time consumed by the CMA-ES algorithm itself, with the same precision tolerance and number of results set for all iterations. CMA-ES should then take approximately the same amount of time across the number of simulations. This explanation becomes clearer when we observe the variation of the number of iterations w.r.t. the number of simulations in Figure 5.9.

Here, the algorithm **Find Border Points** needs around 400 iterations to successfully complete 1,000 simulations (knowing that each iteration grants the algorithm the possibility to look for a scenario "*on*" the border for each output criterion of all three separately). We can also observe the same linear aspect as seen in Figure 5.8, as the optimization process

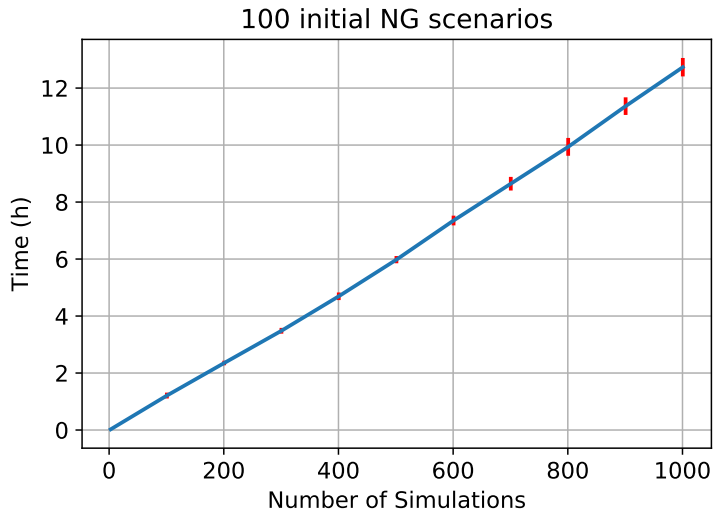

Figure 5.8: Average time (hours) and standard deviations (vertical bars) spent by **Find Border Points** to reach 1,000 simulations for the initial set containing 100 *NG* scenarios.

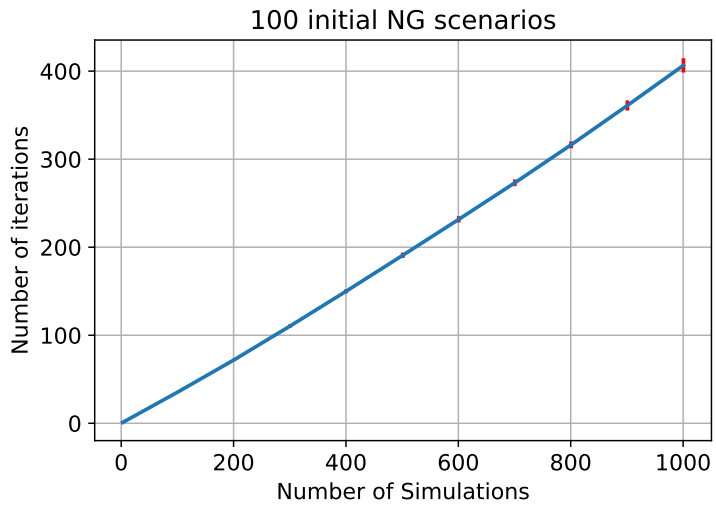

Figure 5.9: Average number of iterations and standard deviations (vertical bars) spent by **Find Border Points** to reach 1,000 simulations for the initial set containing 100 *NG* scenarios.

takes the same amount of time in average. The other initial sets also display this linear curve obtained, and are not presented here.

Up to now, we compared the time needed for each algorithm to reach the same amount of simulations. On one hand, both **Find Border Max** and **Find Border Min** algorithms have a super-linear complexity with respect to the number of simulations, but **Find Border Max** is by far the fastest to complete a thousand simulations, whatever the complexity of the initial set while **Find Border Min** can take up to ten times longer. But in any case,

the complexity of the initial set greatly impacts both algorithms. On the other hand, **Find Border Points** functions differently as its runtime shows instead a linear increase with respect to the number of simulations.

Reduced model accuracy

Before evaluating the number of solutions generated by the three border algorithms, let us take a look at the way the reduced Random Forest model should gradually improve its accuracy along the process.

Figure 5.10 shows the prediction error rate throughout 1,000 simulations for the three border detection algorithms based on a single run on the initial set containing 100 NG scenarios. It is the ratio of the number of times the Random Forest model gave wrong predictions resulting in false G/NG pairs or *off*-the-border scenarios, to the total number of simulations consumed (which is 1,000 here).

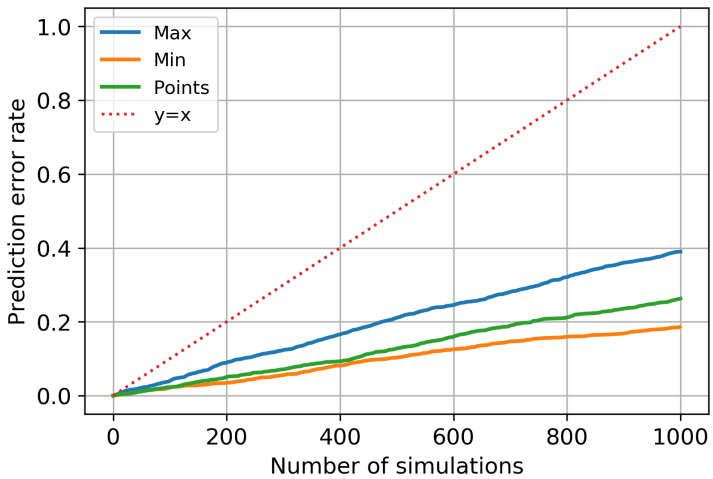

Figure 5.10: Evolution of the prediction error rate related to the Random Forest model based on the initial set containing 100 NG scenarios during 1,000 simulations for the three border detection algorithms.

Similar to Figure 4.4, we compare the curves to the identity function, which corresponds to the worst case of having all Random Forest predictions wrong. We can see that all three border detection curves have a rather sub-linear form throughout the simulations. **Find Border Min** has the best evolution with the least prediction error rates obtained (18.6% at 1,000 simulations), followed by **Find Border Points** (26.3% at 1,000 simulations), and finally **Find Border Max** (39% at 1,000 simulations).

Number of Identified Scenarios

Let us now take a look at the number of solutions found by the three border algorithms, i.e., the number of G/NG pairs detected by **Find Border Max** and **Find Border Min**,

and the number of scenarios "*on*" the border identified by **Find Border Points**.

Let us begin with **Find Border Max**. Figure 5.11 shows the variation of the number of G/NG couples of scenarios detected with respect to the number of simulations. Each graph represents the run of the algorithm for a given initial distribution between the initial sets already seen so far: sets containing 50, 100 and 500 NG scenarios and the input space corners set. Plus, we can visualize the variation of the couples depending on the border for each of the three output criteria of this use case.

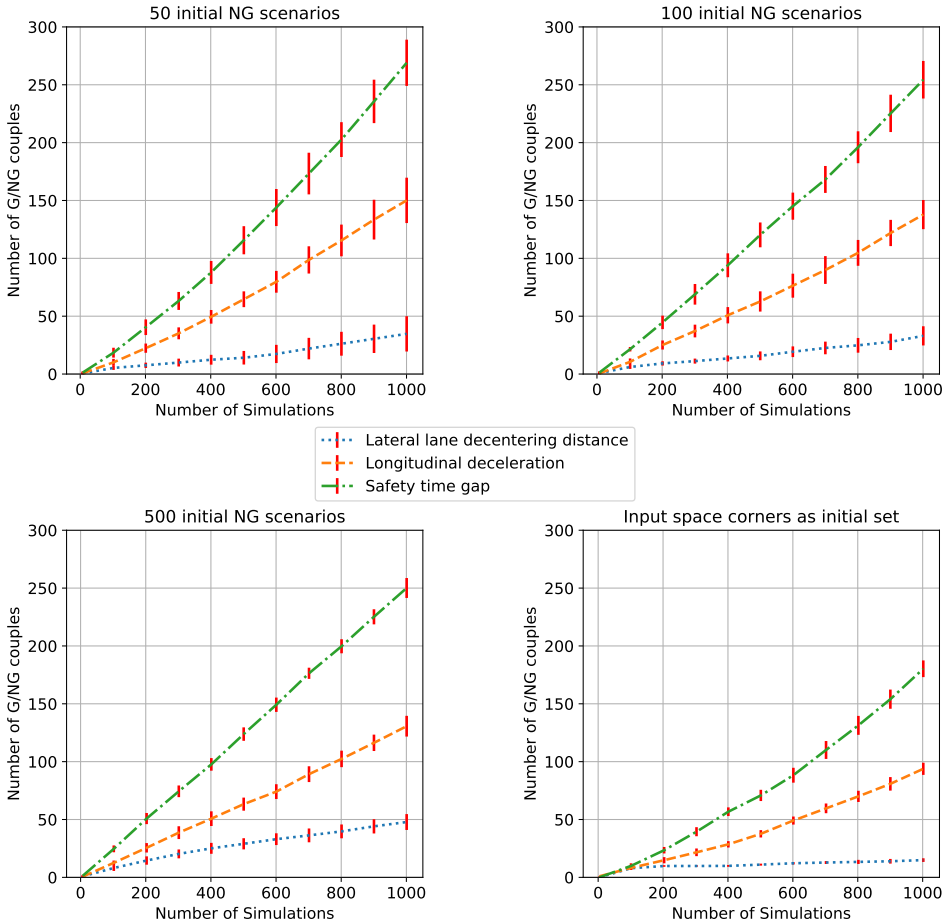

Figure 5.11: Variation of the number of G/NG scenario couples detected by **Find Border Max** for each output criterion throughout the simulations. Each curve is the result of 11 runs with different initial sets containing 50 (top left), 100 (top right) and 500 (bottom left) initial NG scenarios, as well as consisting of the input space corners (bottom right).

We notice that the algorithm manages to detect a different number of couples depending on the output criterion border. It identifies between 200 and 300 couples for the safety time gap warning, between 100 and 150 couples for the longitudinal deceleration warning, and less than 50 couples for the lateral lane decentering distance warning. This can indicate a different border shape for each output warning as the algorithm is conceived to detect couples

in a equal manner among criteria. Plus, if we look at the input space corners case, we can see that the algorithm caught the least number of G/NG couples when compared to the other initial set, despite being the fastest one to achieve 1,000 simulations as seen in Figure 5.3. This can be explained due to the fact that the algorithm proposes multiple new scenarios at once at each iteration because of the biggest maximum distance calculated between the corners. However, the reduced model is still gaining in accuracy during the first iterations as seen in Figure 5.10; a lot of these scenarios proposed initially turned out to be wrong couples when launched by the simulator.

Next, we visualize in Figure 5.12 the variation of the number of G/NG couples of scenarios detected throughout the 1,000 simulations by **Find Border Min** for each output criterion and for the same initial distribution sets as seen before.

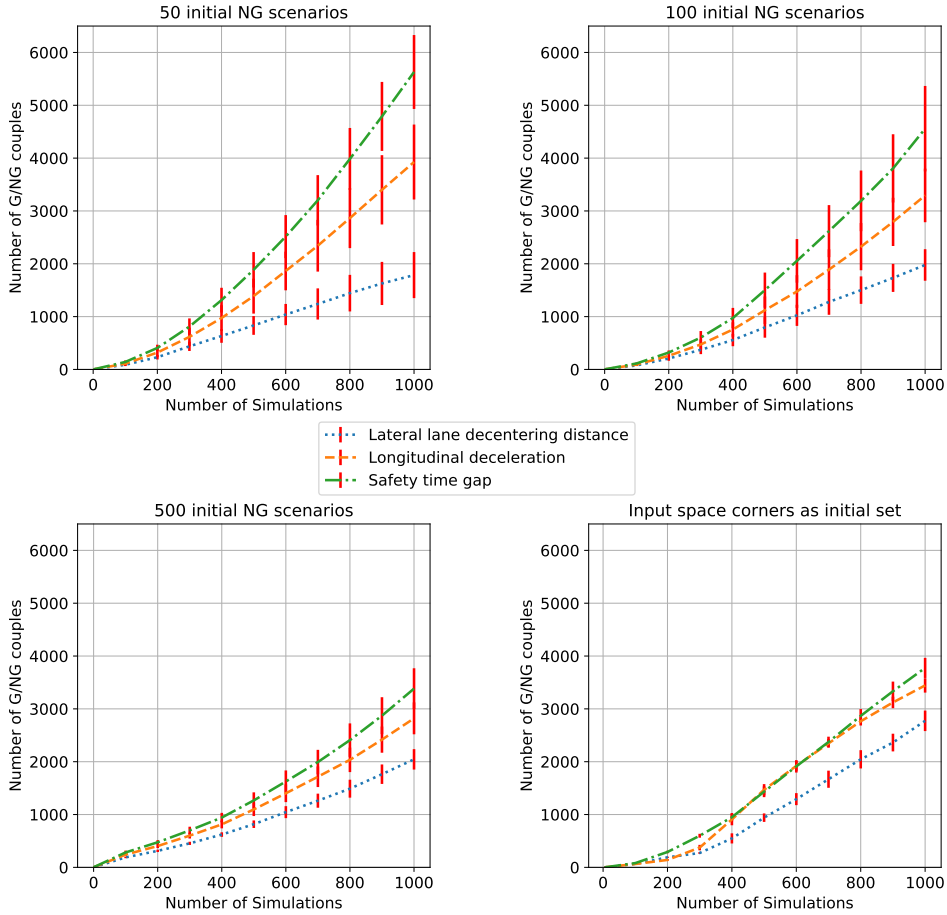

Figure 5.12: Variation of the number of G/NG scenario couples detected by **Find Border Min** for each output criterion throughout the simulations. Each curve is the result of 11 runs with different initial sets containing 50 (top left), 100 (top right) and 500 (bottom left) initial NG scenarios, as well as consisting of the input space corners (bottom right).

Here, the numbers of pairs detected by the **Find Border Min** algorithm is much higher

than the numbers achieved by **Find Border Max**. It managed to detect thousands of couples as opposed to a few hundreds identified by the **Find Border Max** algorithm. For example, the safety time gap criterion spawned between 3,500 and 5,500 couples, while the longitudinal deceleration and lateral lane decentering distance criteria generated between nearly 3,000 and 4,000, and between 2,000 and 3,000 couples respectively. These numbers add up to more than ten times the amounts gathered by the **Find Border Max** algorithm. They also reflect the reasons why **Find Border Min** needs ten times more time to complete the same number of simulations. In fact, these numbers demonstrate that **Find Border Min** detects new pairs among existing scenarios in the archive, i.e., without the need to add new scenarios. Because it is focused on identifying the minimum distance between all existing G/NG scenarios in the archive, it manages to use the existing resources found in the archive to detect more couples. Furthermore, in the case where the initial set is made of the input space corners, the number of discovered pairs is approximately the same than for the other initial sets, while taking much more time to reach the stopping condition: The dichotomy process takes much more time to reach the d_{min} threshold when the distances available between G/NG scenarios are at their maximum values. The evaluation of the results quality will reveal more of the gains produced during the time spent by **Find Border Min** when starting from all these different initial conditions.

Last of all, Figure 5.13 shows the variation of the number of scenarios detected "*on*" the border throughout the 1,000 simulations by **Find Border Points** for each output criterion and for the same initial distribution set of 100 NG scenarios as seen before.

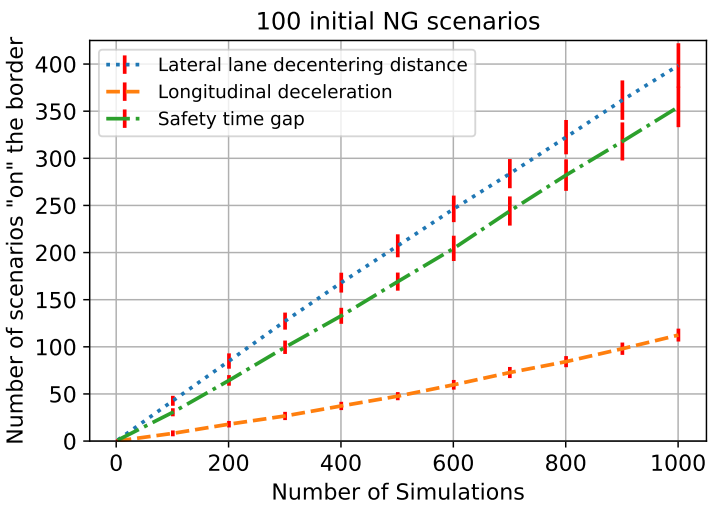

Figure 5.13: Number of scenarios (and standard deviations) detected "*on*" the border by **Find Border Points** for each output criterion throughout the simulations, for the initial set containing 100 NG scenarios.

We can observe that the algorithm was able to detect scenarios "*on*" the border in hundreds, approximately 400, 350 and 110 scenarios were identified for the lateral lane decen-

tering distance, safety time gap and longitudinal deceleration criteria respectively. However, the most noticeable aspect is the predominant numbers found for both continuous criteria when compared to the discrete longitudinal deceleration criterion, and is mainly due to the way we set the border boundaries which define whether a newly simulated scenario can be assimilated "*on*" the border. Plus, these performances are obtained identically for all initial distribution sets, and are therefore not displayed here.

To summarize, all three algorithms managed to detect certain numbers of scenarios as indicated in their respective goals, whether to identify G/NG pairs for the **Find Border Max** and **Find Border Min** algorithms or to propose scenarios located directly "*on*" the border for the **Find Border Points** algorithm. These numbers fluctuate with the initial conditions for the **Find Pairs** algorithms, and are independent of the initial conditions for the **Find Border Points** algorithm. Plus, **Find Border Min** is able to find the highest numbers of pairs among all three algorithms, ranging in the thousands as opposed to hundreds for the other two algorithms. Nonetheless, these numbers are only quantitative assessments, and a more qualitative approach is needed to complement these, and better understand their respective benefits. Next Section is devoted to such qualitative assessment.

5.3.3 Qualitative Results – 1 000 simulations

To better evaluate the quality of the results of the three border detection algorithms, the basic idea was to compare their results to the actual criteria of the same full grid over the search space described in Section 4.5.5: Each normalized input is discretized into 10 values, resulting in a total of 100,000 scenarios for this five-dimensional use case. These scenarios are then simulated to retrieve their real criteria. Next, a metric should be defined in order to assess the quality of the results when compared to the points of the grid. However, the conception of that metric was not as straightforward as for the **Find All Failures** algorithm, and we will now detail the path we followed to come up with some meaningful metric. The initial sets of 50, 100 and 500 NG scenarios are used for quality assessment for all three algorithms. Plus, in addition to the corners initial set for the **Pairs** algorithms, we decided to run the **Find Border Points** algorithm with a fourth initial set containing one single NG scenario to further examine its qualitative performance depending on the initial set.

Precision rate First, we chose to compare the results to the grid by calculating a precision rate. A visual explanation is shown in Figure 5.14, as we will describe more thoroughly the different steps in computing the metric while referring to the figure for better understanding.

Basically, the left grid in the figure shows a simplified 2-D shape of the border, which separates the G scenarios in green circles on the left side of the border from the NG scenarios in red circles on its right side. The middle grid shows the introduction of the G/NG couples illustrated as green and red triangles respectively. The idea is then to take each grid scenario, i.e. each circle, and to look for its nearest neighbor between the algorithm scenarios, i.e. the triangles. Their criteria evaluations are then compared to check if they have both the same

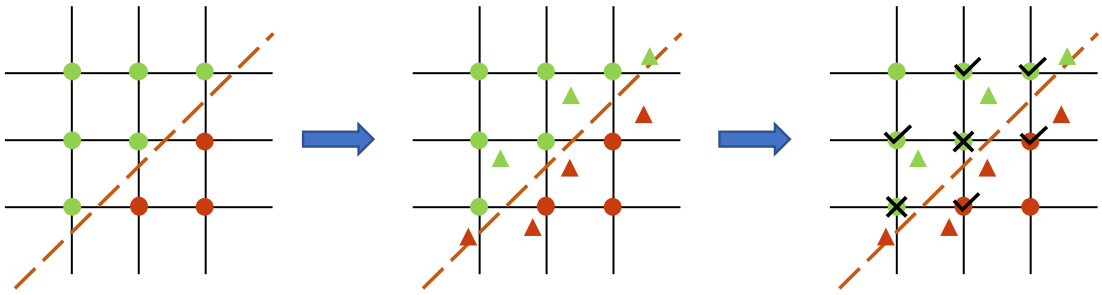

Figure 5.14: Visual explanation of the **precision rate**: first metric when comparing to the grid.

evaluations, which should indicate that the scenarios proposed by the algorithm succeeded in replacing a grid of 100,000 scenarios of the same search space with a few thousands that delimit the G/NG border. Thus, the precision rate metric is computed as the ratio of all the occurrences of equal criteria evaluations between the grid scenarios and their nearest algorithm-produced neighbors to the total number of grid scenarios. A first attempt of computing this metric is shown in Figure 5.15 where it is applied on **Find Border Max** for all the initial distribution sets.

We can observe from these curves that the precision rates reach systematically high values from the very start, as all curves mark rates higher than 75% from their initial sets before any algorithm interference. Then, the curves tend to increase very slowly across the simulations to a few gain percentages for the initial sets containing 50 and 100 NG scenarios, whereas the 500 NG initial set has near constant curve shapes. Additionally, the input space corners set shows the biggest boost in rates across the simulations where the rates increase by a 10% add value.

However, these curves shapes illustrate clearly that a good precision rate is already attained before any optimization, pointing out to fairly acceptable distributions of the initial sets. This would also explain the small increases along the iterations while adding new scenarios to the archive, especially when we are comparing 100,000 grid scenarios to mere hundreds of scenarios algorithms.

Therefore, we decided to take into account not only the nearest algorithm scenario neighbor to each grid scenario, but the 3 or 5 nearest neighbors instead. The average of their evaluations is then computed before comparing it to the grid scenario evaluations. In that way, we try to give more robustness to the precision rate metric and decrease its instability in being very dependent of the nearest neighbors only. Figure 5.16 shows the different curves resulting in taking into consideration 3 and 5 nearest neighbors for the computing of the precision rate on the **Find Border Max** algorithm results.

We can observe that the resulting curves bear small differences when compared to the

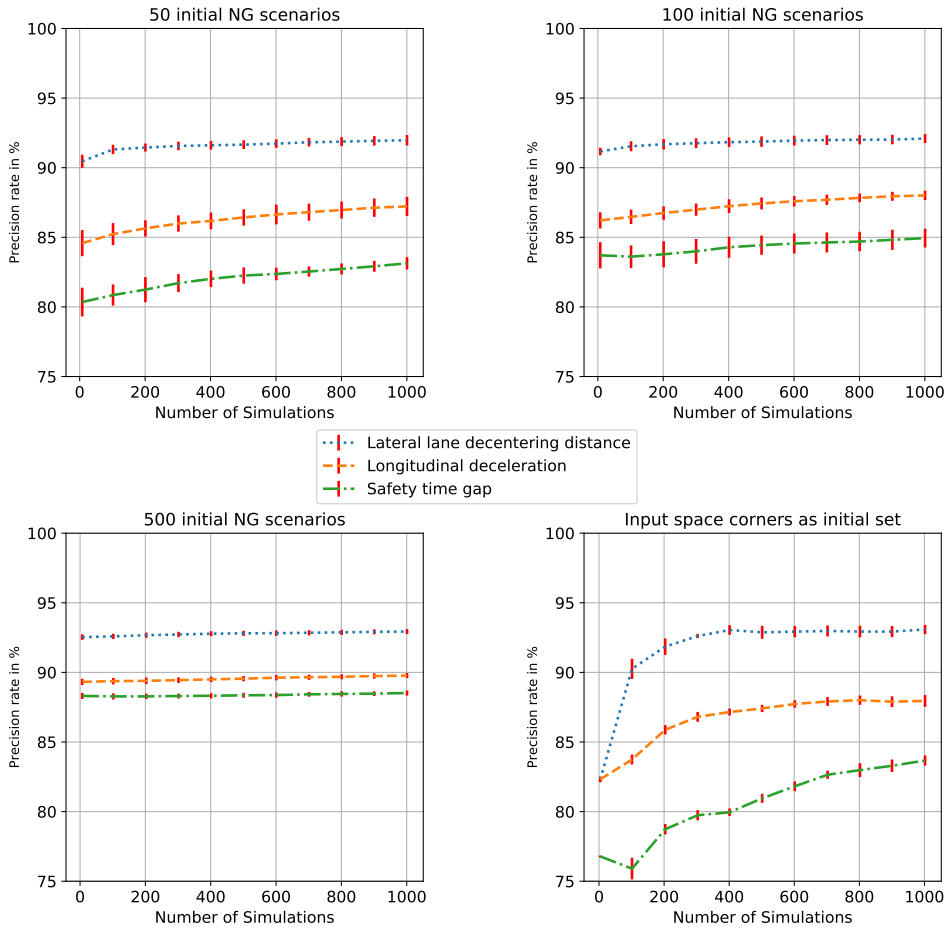

Figure 5.15: Variation of the **precision rate** metric applied on the **Find Border Max** algorithm throughout the simulations. Each curve is the result of 11 runs with different initial sets containing 50 (top left), 100 (top right) and 500 (bottom left) initial *NG* scenarios, as well as consisting of the input space corners (bottom right).

nearest neighbor curves in Figure 5.15. The only noticeable difference is the precision rate values at the beginning of the algorithms which gives more approval to the impact of the initial distribution set on the computing of this metric. Plus, we can observe some decreases that occurred when introducing multiple neighbors and adding new scenarios. This could be explained by cases where the nearest neighbors are more or less distant before other closer points of different evaluation are added. The same visualizations are noticed for the **Find Border Min** and **Find Border Points** algorithms and will not be presented. Thus, taking into account 3 or 5 nearest neighbors turned out to be ineffective in resolving the main problem with this metric in efficiently assessing the quality of the results obtained. This is why we decided to elaborate an other metric: the border rate.

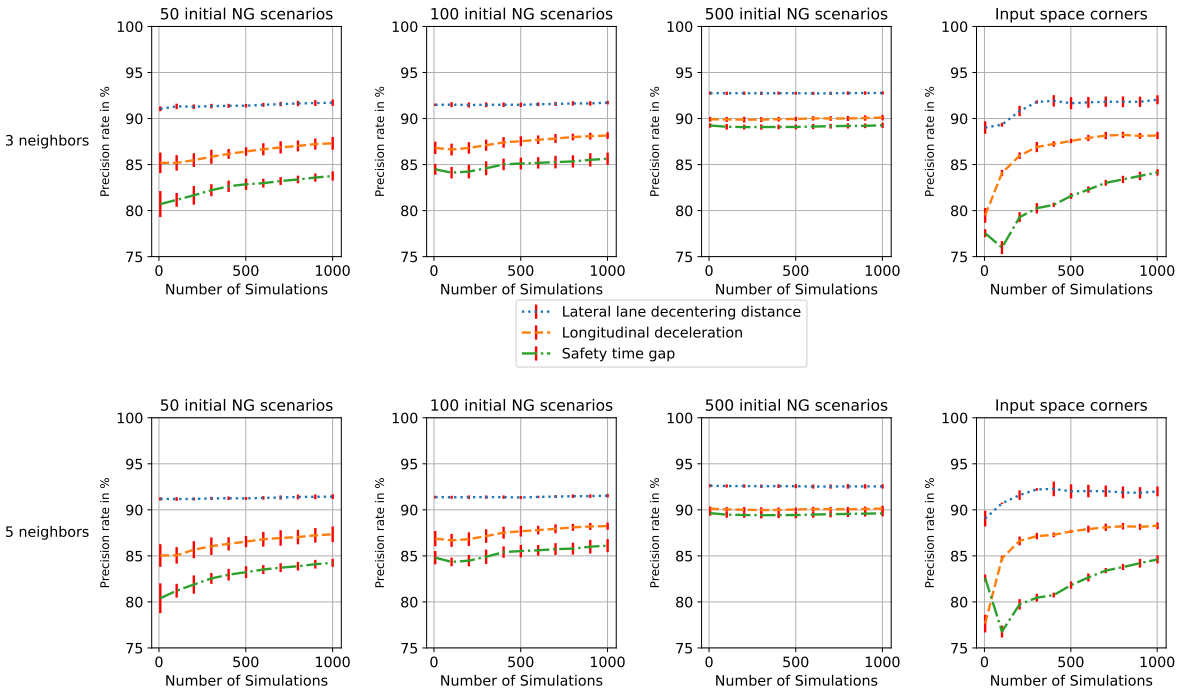

Figure 5.16: Variation of the **precision rate** metric applied on the **Find Border Max** algorithm throughout the simulations by taking into account the 3 and 5 nearest neighbors between the algorithm scenarios. Each curve is the result of 11 runs with different initial sets containing, from left to right, 50, 100, and 500 initial NG scenarios, as well the input space corners.

Border rate Similarly to the precision rate, we will explain the process of computing the border rate metric while referring to grid visuals, which are shown in Figure 5.17.

The first visual (top left) resumes the standard green and red circle scenarios separated by the border line to the left and right sides respectively. The idea is to detect the nearest G/NG couples of scenarios in the grid as shown in the second visual (bottom left). These grid scenario couples can be seen as a discrete and simplified way of perceiving the border, since it should naturally be located between all these close G/NG grid couples. Then, we introduce the algorithm scenarios shown as green and red triangles to differentiate the G and NG evaluations in the third visual (bottom right). Next, each grid scenario has its evaluation criteria modified according to its nearest neighbor among the algorithm scenarios. We also take into account the impact of considering multiple nearest neighbors too. Finally, the G/NG couples of grid scenarios obtained previously are re-evaluated to check if they persisted to be scenarios of different evaluations, as seen in the fourth visual (top right). Thus, the border rate is the ratio of the grid couples that remained with different criteria evaluations after the introduction of the algorithm scenarios, to all the grid couples detected before the criteria evaluation update due to the algorithm scenarios. This metric has been

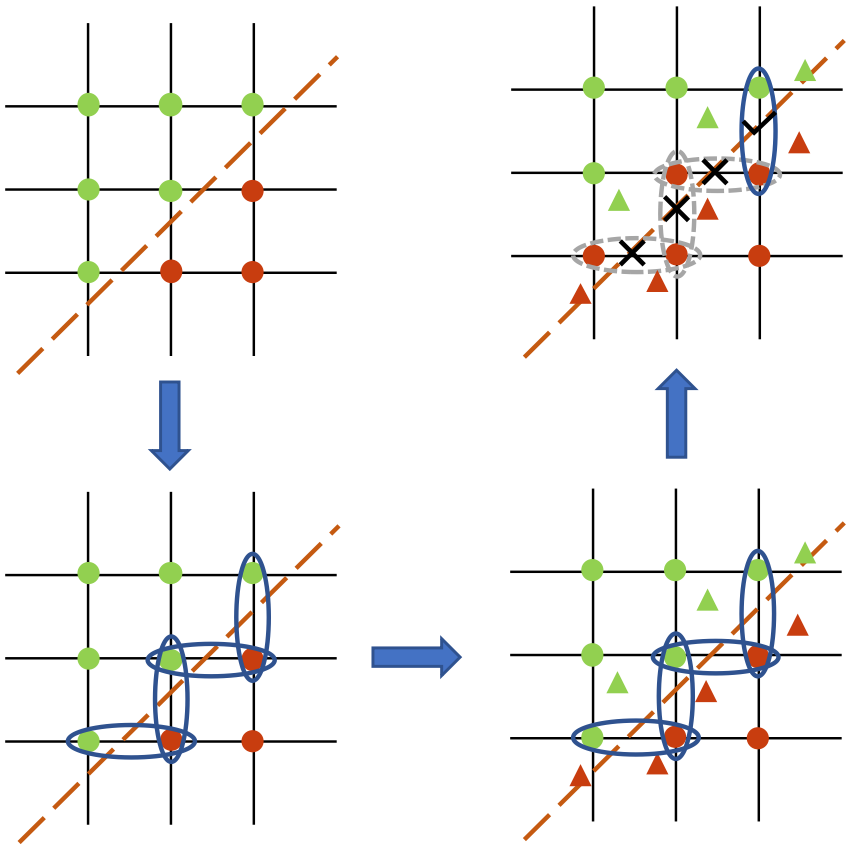

Figure 5.17: Visual explanation of the **border rate**: second metric when comparing to the grid.

computed for the three algorithms for all the initial distribution sets considered thus far. In particular, Figure 5.18 displays the variation of the border rates implemented for the **Find Border Points** algorithm when taking into consideration the nearest scenario algorithm neighbors, as well as the nearest 3 and 5 neighbors, for initial sets containing 1, 50, 100 and 500 *NG* scenarios respectively.

We notice that the curves obtained show mediocre results overall. For example, all rates attain a maximum of 30% to 35% across all considered cases for all criteria. The computation at zero simulation checkpoint shows the impact of the initial set on the results, where it is at its lowest value for the initial set of 1 *NG* scenario hitting nearly 0% for all criteria, and steadily increases when enlarging the initial set, e.g. between 5% to 15%, 5% to 20%, and 15% to 30% for the initial sets containing 50, 100 and 500 *NG* scenarios respectively. Plus, the rates increase across the simulations remains little, the only possible exception being the single scenario initial set since it begins with 0% and manages to grow to around 20% for

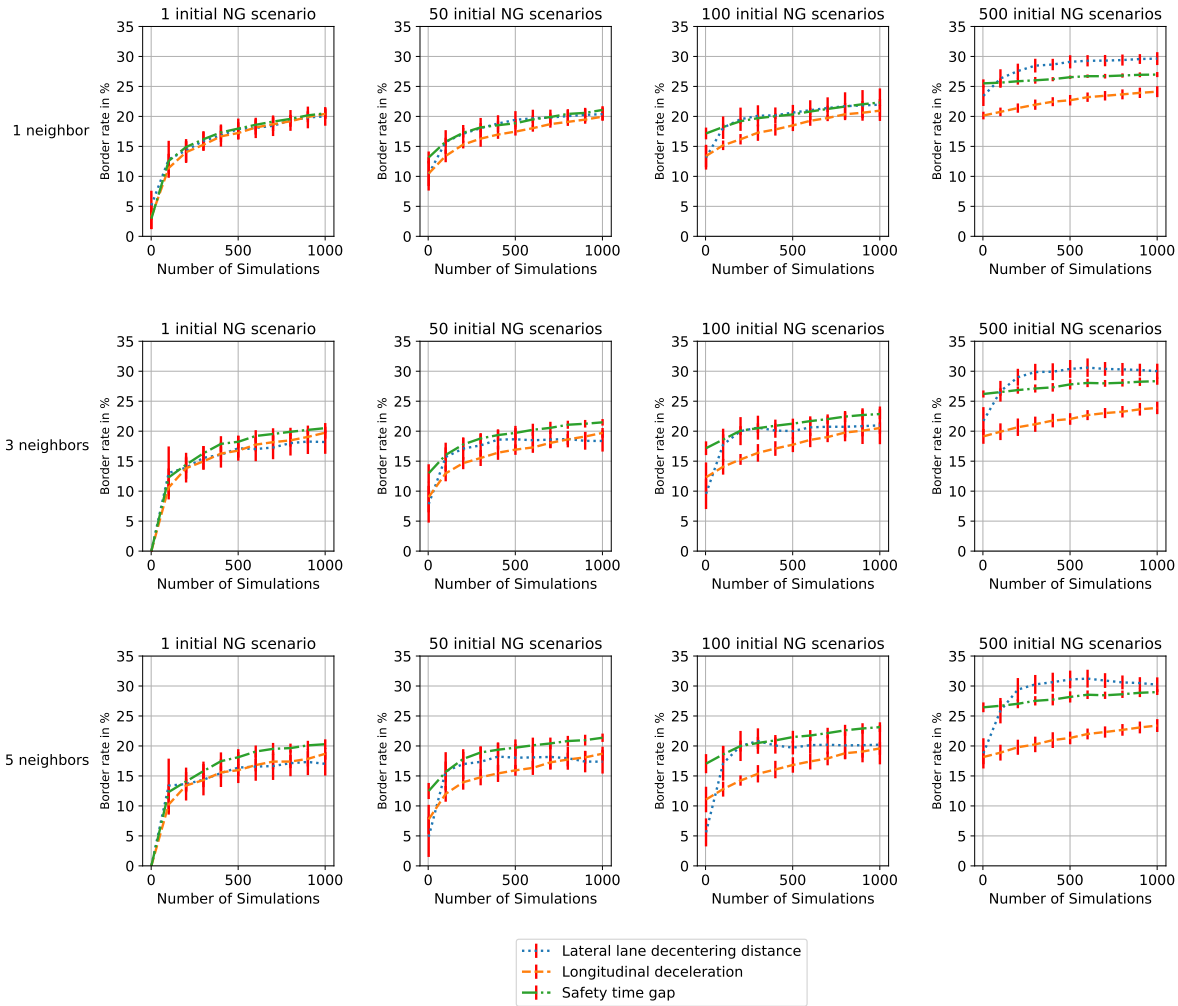

Figure 5.18: Variation of the **border rate** metric applied on the **Find Border Points** algorithm throughout the simulations by taking into account the nearest 1, 3 and 5 neighbors between the algorithm scenarios. Each curve is the result of 11 runs with different initial sets containing, from left to right, 1, 50, 100, and 500 initial *NG* scenarios.

all criteria. These results are highly similar to the ones obtained with the other two **Find Border Max** and **Find Border Min** algorithms, and are therefore not presented.

While achieving mediocre border rates can be seen as a contradiction to the relatively high accuracy rates noted previously, the core problem could stem from the fact that we have not taken into account in these calculations the case where the detected border by the algorithm is simply slightly offset: this could give a border rate of 0% here. For instance, if we look back at the last visual of Figure 5.17, we can see how a slightly offset border detected by the algorithm could very well go unnoticed by the border rates calculations. The resulting visual is shown in Figure 5.19, where the right grid shows the offset of the detected border

after having updated the criteria evaluations of the grid scenarios. The original grid border is the orange dashed line, whereas the resulting border, which passes between all the G/NG grid couples with newly updated criteria evaluations, is represented as the blue dash-dotted line. Hence, the border rates are limited in taking into account a possible offset in the border detection by the algorithms.

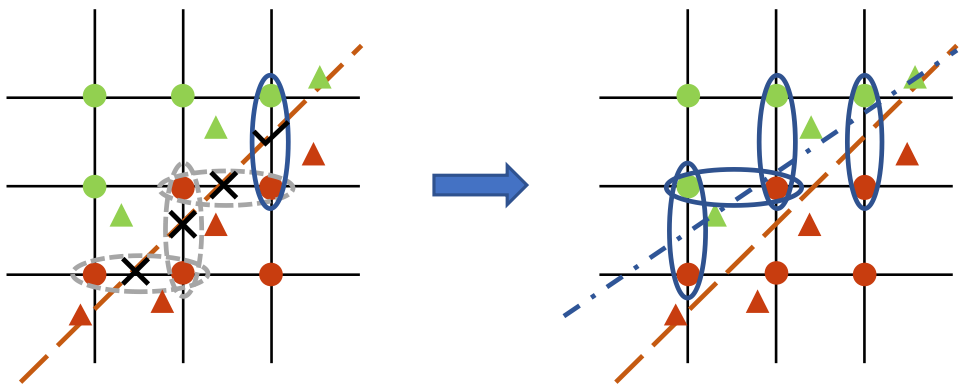

Figure 5.19: Slight offset of the border detected by the algorithm unnoticeable by computing **border rates** when comparing to the grid.

Thus, because of this limitation in the border rates results, it would be more interesting in this case to determine whether we can manage to find the same NG areas with the algorithms compared to the NG areas without any algorithm action, and to calculate the distance error between the areas which quantifies the border detection offset. This is why we finally adopted this third and final metric: connected components.

Connected components Figure 5.20 illustrates the procedure of computing the connected components metric when compared to the grid to better evaluate the performances quality of these algorithms.

First of all, we retrieve the same grid of scenarios across the input search space, represented as green and red circles referencing their respective evaluated output criteria G/NG separated by the dashed border at the left and right side respectively (top left). Then, we calculate the connected components for all NG scenarios by iterating over them all as visualized by the right-angled orange triangle (bottom left). The idea is to determine areas of nearest neighbors among the NG scenarios. For instance, when coming across a NG scenario, we look for its nearest neighbors if they characterize as NG scenarios too. If that turned out to be the case, we take the NG scenarios detected into account into the connected component area, and we move to these newly detected scenarios and check for their nearest neighbors for NG scenarios, and so on until iterating over all the NG grid scenarios. If all nearest neighbors

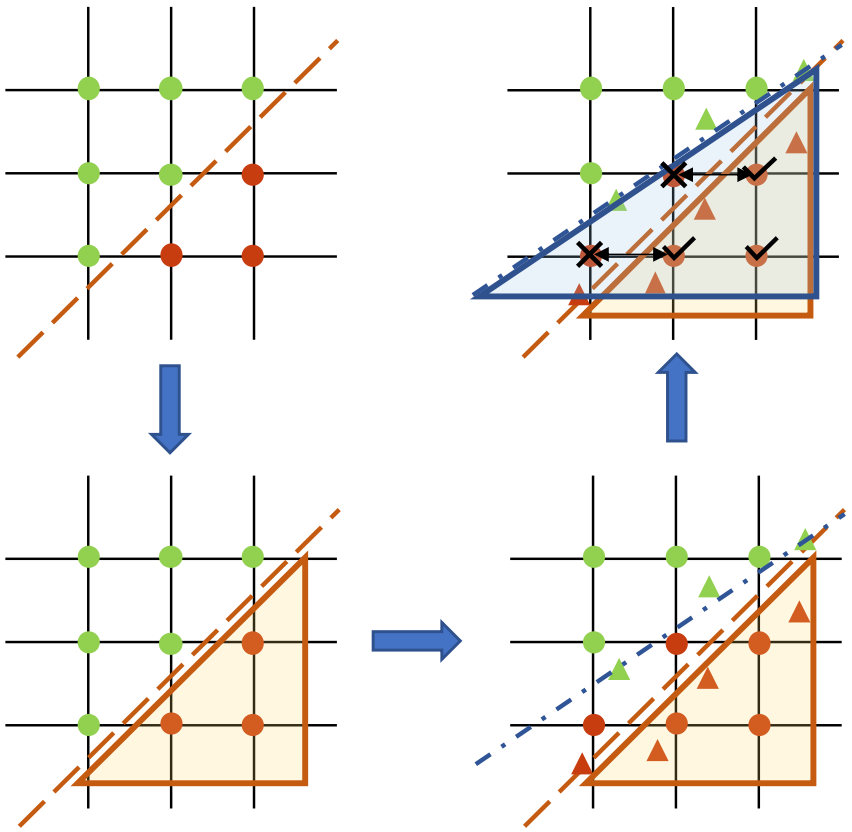

Figure 5.20: Visual explanation of the **connected components**: third and final metric when comparing to the grid.

of a particular NG scenario are G scenarios, then the connected component area is stopped at that scenario, and a new connected component is created when we resume iterating over the grid scenarios and stumble upon another NG scenario.

Next, we introduce the scenarios generated by the algorithm and represented by green and red triangles following their criteria evaluations, and we modify the grid scenarios evaluations by taking into consideration their nearest neighbors among all algorithm-produced scenarios as done previously (bottom right). We can also see the supposedly dash-dotted blue border detected by the algorithm with the slight offset. Finally, we re-calculate the connected components of the grid scenarios with the newly updated criteria evaluations (top right). We should obtain different NG areas as visualized by the new right-angled blue triangle. Here, we detect, on one hand, which NG grid scenarios were identified by the new connected components and were effectively part of the original connected components before changing the evaluations criteria, and on the other hand, which of these NG scenarios in the new connected components were originally G scenarios in the original connected components.

These last scenarios are therefore classified incorrectly by the algorithm and represent the offset made by the algorithm. We then compute, on one hand, the ***NG* classification rate**, which is the ratio of the *NG* grid scenarios found in the new connected components that are indeed *NG*, to the total number of *NG* scenarios found in the original connected components before algorithm action. On the other hand, we calculate the **offset distances**, which are the distances between all the scenarios incorrectly classified as *NG* by the algorithm and the original connected components to check whether this offset error is acceptable or not. Additionally, we are interested in calculating the **coverage distances**, which are the distances between the *NG* scenarios of the original connected components that remained undetected by the algorithm, and the *NG* scenarios of the updated connected components that were rightfully classified as *NG* by the algorithm, to see if the algorithm is able to reach out to *NG* scenarios everywhere in the search space.

***NG* classification rate** In this paragraph, we will begin tackling all algorithm results to finally be able to evaluate and compare their qualitative performances. First, Figure 5.21 shows the evolution of the *NG* classification rate for the **Find Border Max** algorithm for all criteria evaluations across the simulations for the four initial sets considered thus far, as well as taking into account the nearest 1, 3 and 5 neighbors among the algorithm scenarios when updating the grid evaluations.

We begin by noticing the impact of the initial set on the end results, as the rates tend to increase with the size of the initial set when comparing the distributions containing 50, 100 and 500 *NG* scenarios. Plus, the trends seem to differ depending on the criterion, as the longitudinal deceleration and safety time gap rates slowly increase with the simulations overall, whereas the lateral lane decentering distance rates stay constant or even show a slight decrease. This can be explained due to the fact that **Find Border Max** was unable to introduce as many scenarios for the lateral lane criterion than for the other two criteria as seen in Figure 5.11. This decrease is notably observed in the second half of simulations launched for the input space corners. Besides, taking into consideration more neighbors also slightly decrease the overall trends at the beginning of the optimization, but its effects tend to mitigate along the iterations as the reduced model gains in accuracy. Finally, the final rates obtained are acceptable for the longitudinal deceleration and safety time gap rates (around 60% to 70% and 70% to 80% respectively), but fell short for the lateral lane decentering distance criterion with rates between 10% to 50%.

Next, Figure 5.22 illustrates the variation of the *NG* classification rate for the **Find Border Min** algorithm for all criteria evaluations across the simulations with the same experiments conducted for the **Find Border Max** algorithm.

Some observations in the previous figure are also seen here, e.g. the rates increase with the size of the initial set, and their decrease with the number of nearest neighbors. However, although the trends remain to be dependent on the criterion evaluated, the rates either show a decrease for the lateral lane criterion, or a constant rate throughout the simulations for the

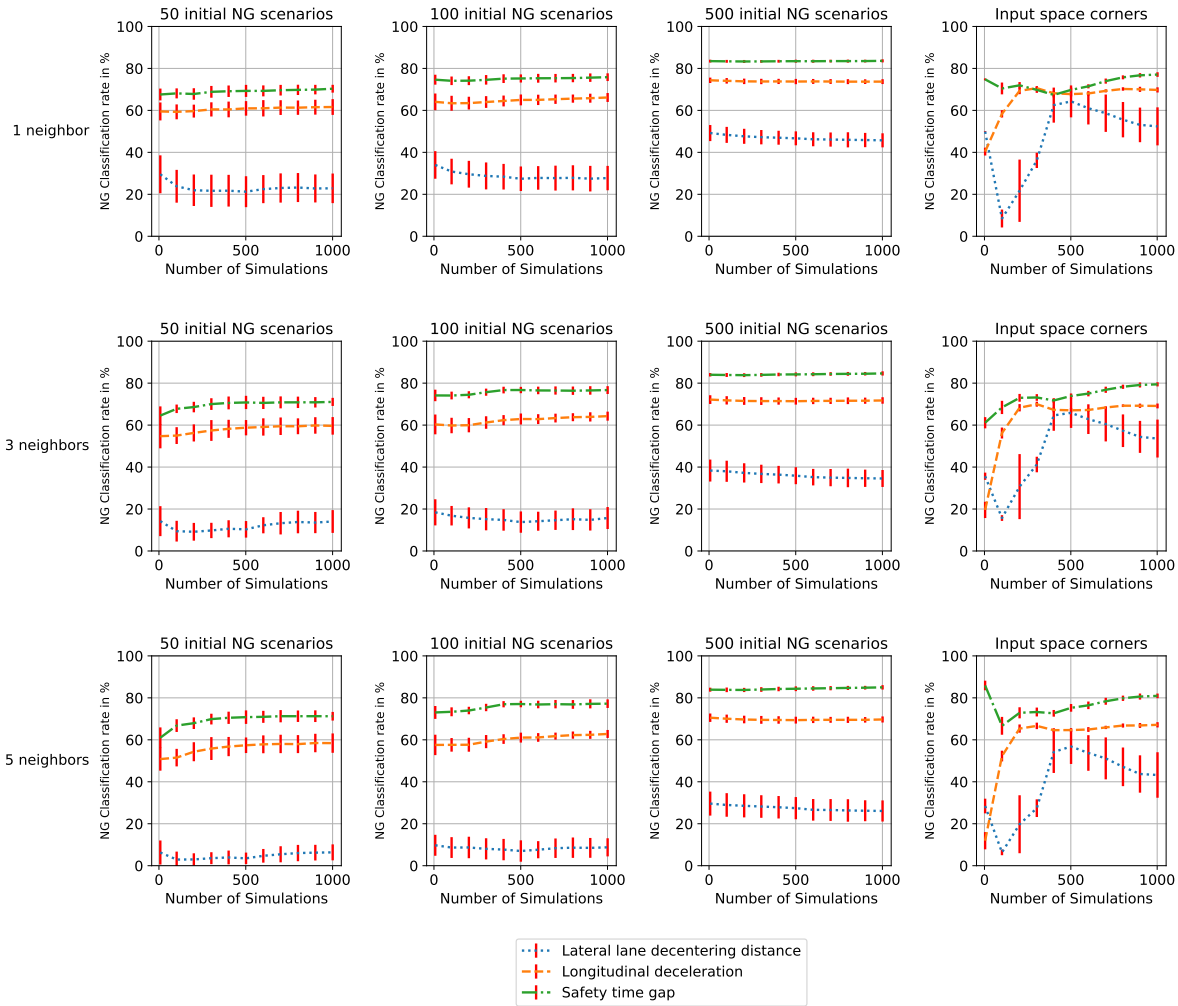

Figure 5.21: Variation of the *NG* classification rate related to the **connected components** metric applied on the **Find Border Max** algorithm throughout the simulations by taking into account the nearest 1, 3 and 5 neighbors between the algorithm scenarios. Each curve is the result of 11 runs with different initial sets containing, from left to right, 50, 100, and 500 initial *NG* scenarios as well as the input space corners.

remaining two criteria. Nevertheless, the input space corners boasts high rates increase for the longitudinal deceleration making it the notable exception, which gives more worth to its longest running time between all experiments. The final rates obtained remain acceptable for the longitudinal deceleration and safety time gap rates (around 60% to 80% and 70% to 90% respectively), and mediocre for the lateral lane decentering distance criterion with rates between 10% to 50%. However, both **Find Border Max** and **Find Border Min** algorithms unveil small variation rates throughout the simulations, which can be explained due to their incapacity in exploring outside their initial sets.

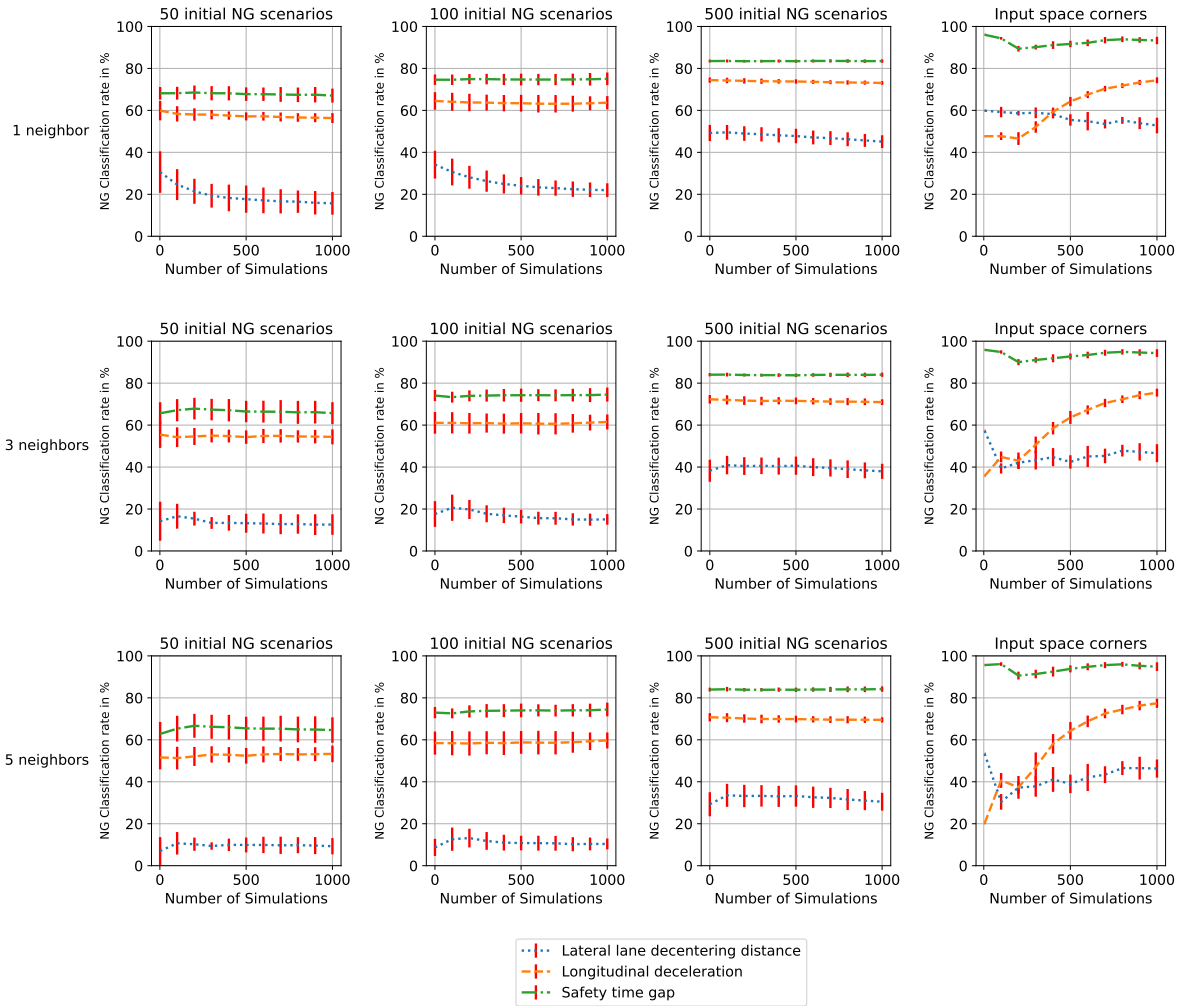

Figure 5.22: Variation of the *NG* classification rate related to the **connected components** metric applied on the **Find Border Min** algorithm throughout the simulations by taking into account the nearest 1, 3 and 5 neighbors between the algorithm scenarios. Each curve is the result of 11 runs with different initial sets containing, from left to right, 50, 100, and 500 initial *NG* scenarios as well as the input space corners.

Finally, Figure 5.23 shows the variation of the *NG* classification rate for the **Find Border Points** algorithm for all criteria evaluations throughout the simulations.

The main difference observed in this figure is that all curves show rates increase for all initial distribution sets and for all criteria, especially the lateral lane decentering distance criterion. For instance, even though increasing the nearest neighbors number still impacts the rates at the beginning of the iterations, the rates for the lateral lane criterion boast high increases going from around 20% to more than 80% as seen in particular for the initial sets of 50 and 100 *NG* scenarios. Plus, we can notice that there is some stability in the results when

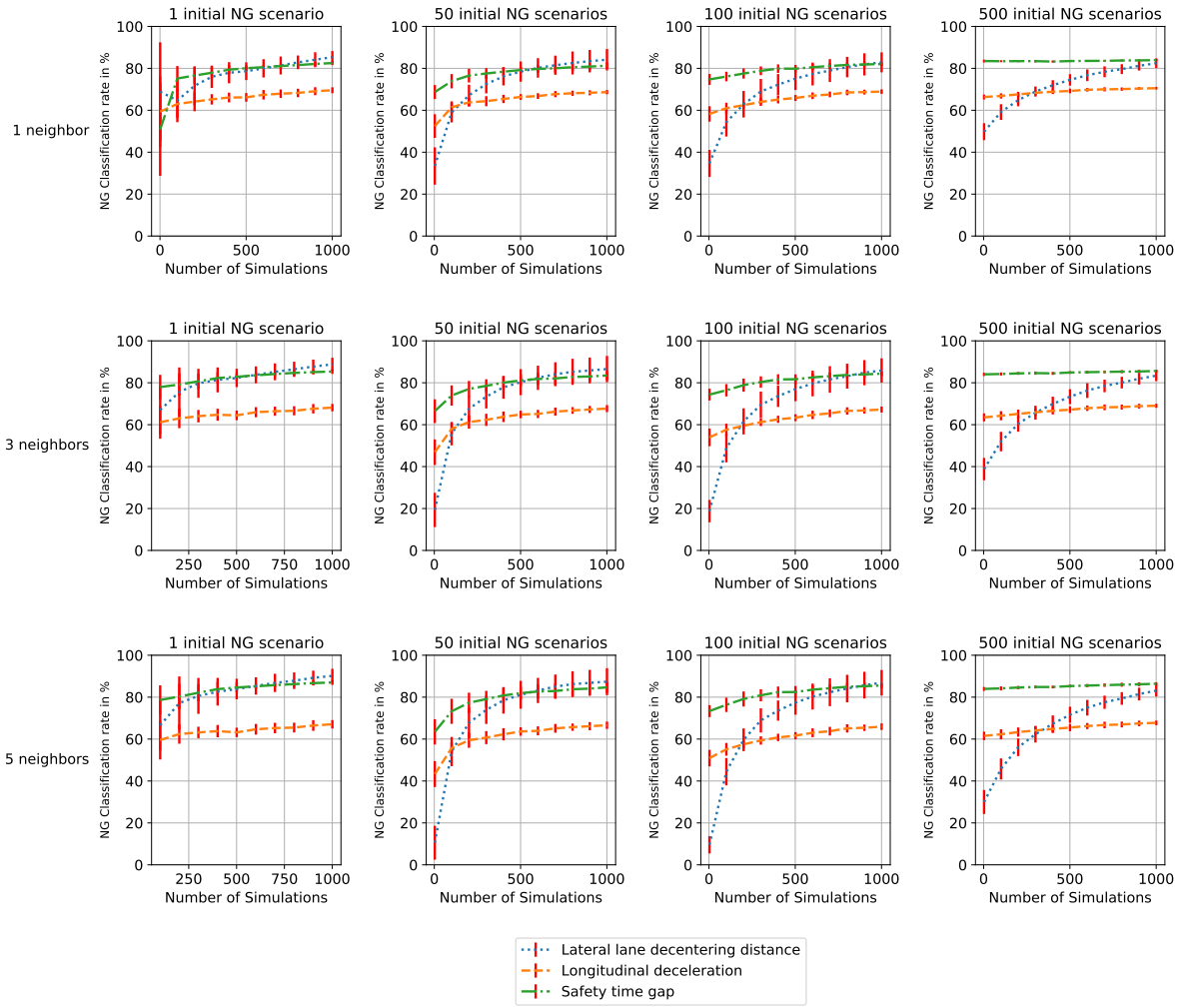

Figure 5.23: Variation of the *NG* classification rate related to the **connected components** metric applied on the **Find Border Points** algorithm throughout the simulations by taking into account the nearest 1, 3 and 5 neighbors between the algorithm scenarios. Each curve is the result of 11 runs with different initial sets containing, from left to right, 1, 50, 100, and 500 initial *NG* scenarios.

changing the initial set, as the algorithm always manages to increase these rates. Besides, the final rates obtained are deemed acceptable for all three criteria, as the lateral lane decentering distance, the longitudinal deceleration and safety time gap rates attain between 80% and 90%, 60% and 70%, and 80% to 90% respectively. The lower final rates achieved for the longitudinal deceleration criterion can be caused by the complexity in detecting "*on*"-the-border scenarios for a discrete criterion, but still remain acceptable overall when compared with the other two algorithms. Next, we are going to delve deeper in the quality evaluation by analyzing the relevant distances between the original grid connected components and the

new ones generated by the algorithms.

Distances statistics We will begin by looking at the **offset distances**, i.e., the distances that separate the original grid connected components and the *NG* scenarios found in the updated connected components but that turned out to be incorrectly evaluated as *NG*. Since we have observed thus far that the number of nearest neighbors between the algorithm scenarios that update the grid evaluations had a small impact on the results, we decided to look only at the one nearest neighbor case, and combine the results of all the algorithms directly in one single figure: Figure 5.24 shows the evolution of the mean offset distances computed between the original connected components and the incorrect *NG* scenarios of the updated connected components for all criteria, for all three algorithms and for all the initial distribution sets.

The curves tend to decrease along the iterations, with values attaining less than 0.2 for all algorithms and all initial conditions. This can be explained due to the fact that, with each new simulated scenario added, the reduced model is gaining in accuracy. Therefore, the offset in the border detection will decrease along the iterations. The only notable exception is the input space corners for **Find Border Min**, which suggests that it would need even more iterations for the offset to begin to shrink, knowing that this experiment was the most costly in running time. As for the other experiments, all algorithms tend to reduce this offset throughout the simulations. We also notice that all these curves seem to approach the value equal to 0.11. This value corresponds to the minimal precision distance d_{min} fixed for the three algorithms, knowing that all three of them were restricted to propose new scenarios that only respected this distance with the remaining scenarios of the archive. It also is equal to the minimal distance value found between nearest neighbors of the grid, as we have a grid of 100,000 scenarios with 5 continuous inputs broken into 10 values each (along with the intervals limits). Hence, if the results of these mean distances approach this particular value, this means that all algorithms manage to shrink the offset in their border detection as much as possible while following this same minimum precision distance, deeming the offset error as acceptable.

However, in order to better understand the results obtained with the *NG* classification rate, we should consider looking at other relevant distances too. Figure 5.25 shows the evolution of the mean **coverage distances**, i.e. the distances computed between real *NG* scenarios of the updated connected components, and the undetected *NG* scenarios of the original connected components. Similar to the previous figure, the results are shown for all three algorithms and for all the initial distributions set while considering the nearest neighbor when updating the grid evaluations.

While the previous mean distances seen in Figure 5.24 represent the offset in the border detection, these mean distances shown correspond to the capacity of the algorithm in exploring all the *NG* scenarios of the original grid connected components. If these distances decrease with the iterations, it means that the algorithm is gradually detecting the original connected

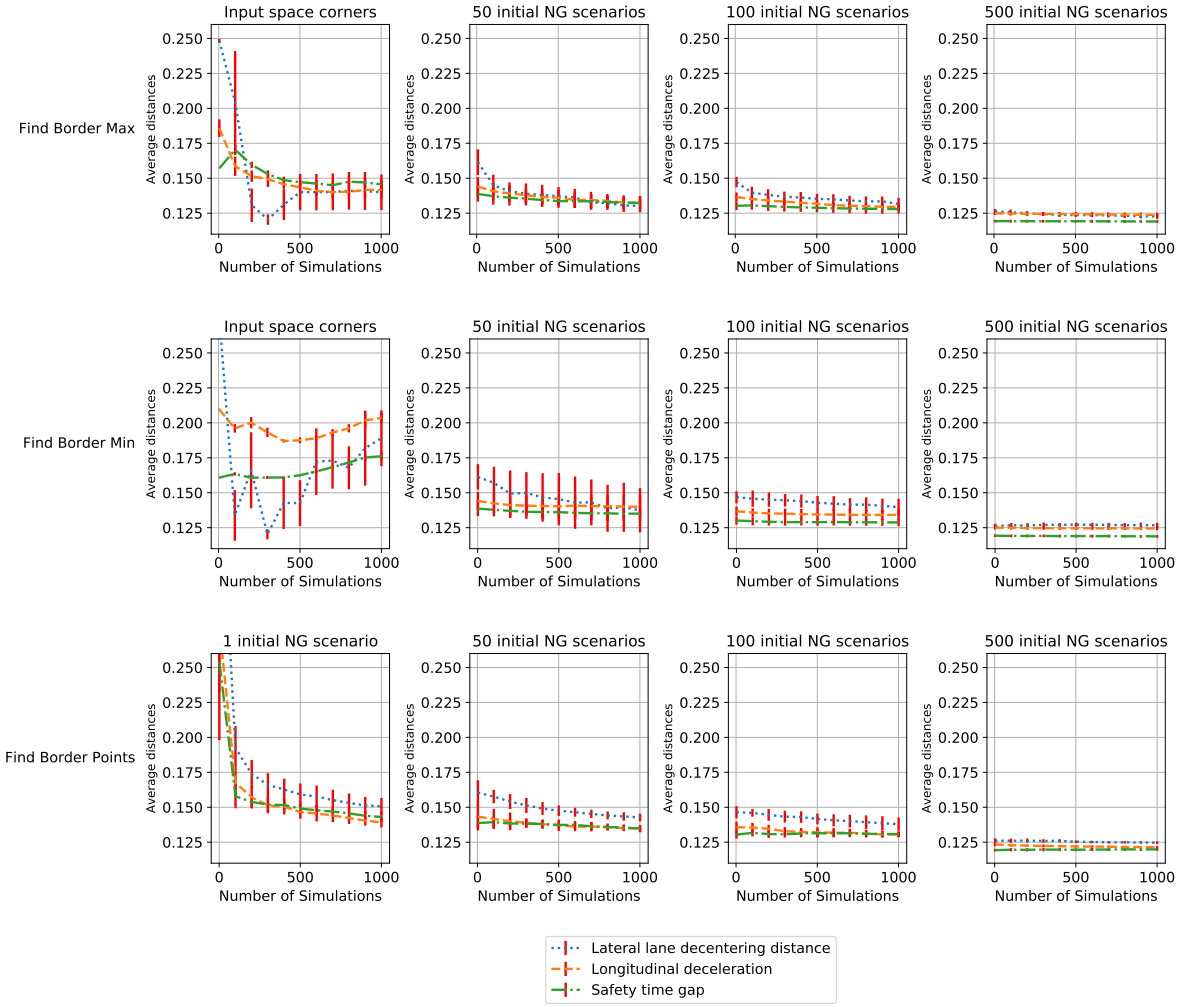

Figure 5.24: Mean *offset distances* (and standard deviations) between the original *NG* **connected components** and the *NG* scenarios of the updated connected components that were incorrectly classified as *NG* by **Find Border Max**, **Find Border Min** and **Find Border Points** for the different initial sets.

components. We can observe differences in the results between the algorithms and between the initial distributions sets. For instance, if the initial set is big enough, e.g. the 500 *NG* initial set, these distances have always low values which are already close to the minimal precision distance set even without any algorithm action. As we reduce the initial set size, e.g., the 50 and 100 *NG* initial sets, we notice that the curves for **Find Border Min** and **Find Border Max** have a constant shape and do not illustrate a notable decrease throughout all the simulations. Plus, the results related to the lateral lane decentering distance criterion correspond to the highest average distances values ranging between 0.2 and 0.25. Diversely, the **Find Border Points** algorithm manages to keep on decreasing these distances across

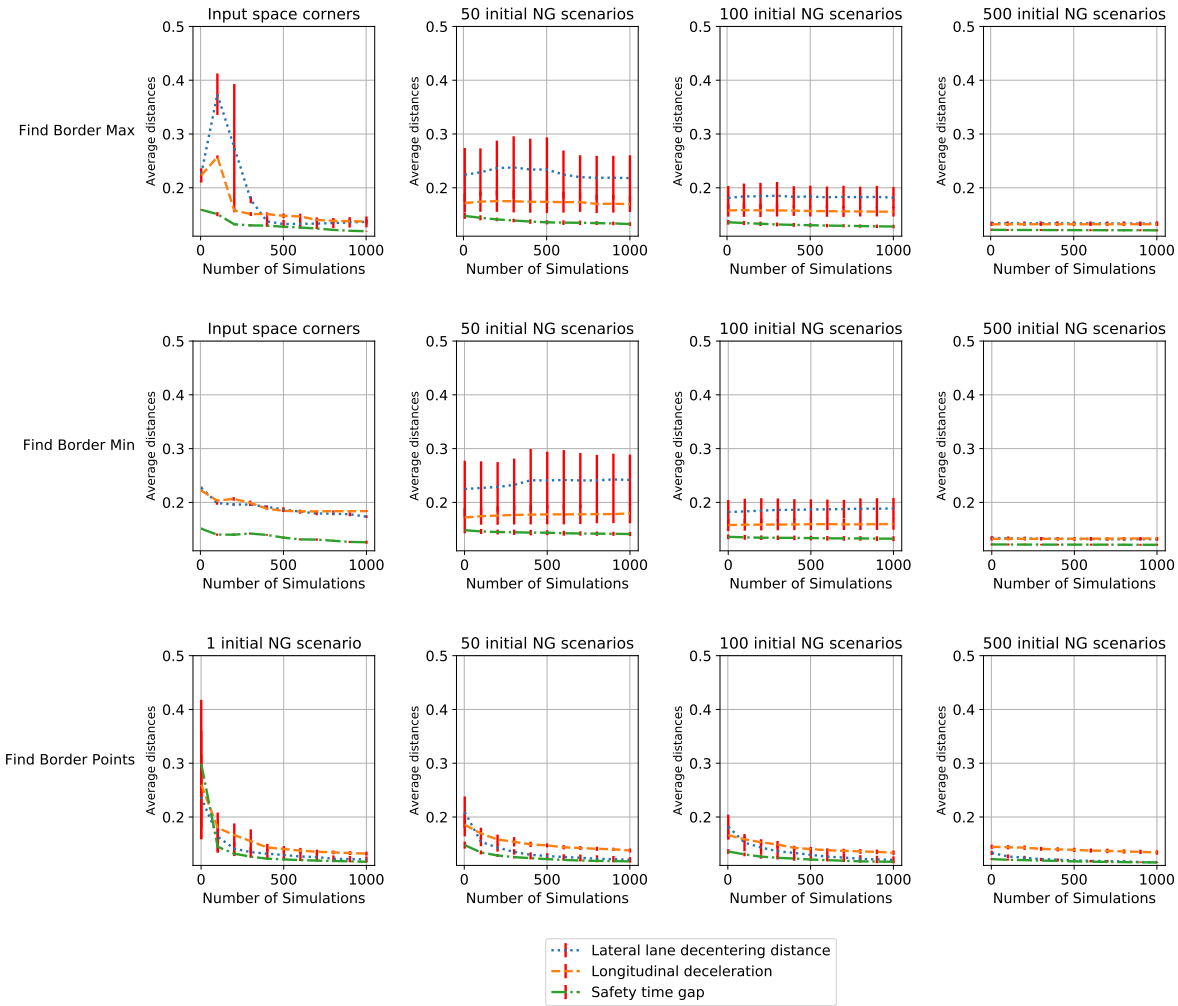

Figure 5.25: Mean *coverage distances* (and standard deviations) between the undetected *NG* scenarios of the original **connected components** and the *NG* scenarios of the updated connected components that were correctly classified as *NG* by **Find Border Max**, **Find Border Min** and **Find Border Points** throughout the simulations. Same initial distribution sets are considered for each algorithm.

the simulations for these same initial sets and for all criteria while approaching the minimal precision distance value. It is even able to reach this value with only a single initial *NG* scenario in its initial set. As for the input space corners set, **Find Border Max** is able to explore the connected components because of the accessibility of the whole input space and its ability in suggesting multiple scenarios at each iteration. On the contrary, **Find Border Min** is far from achieving these same results with just a thousand simulations, knowing that it took around 80 hours to reach these results.

All these results regarding distances give some insights regarding the differences seen in

the *NG* classification rates results in the previous figures. **Find Border Points** achieved the highest percentages, whereas the **Find Border Max** and **Find Border Min** algorithms results were much more modest, especially for the lateral lane decentering distance criterion. Additionally, the larger the initial set, the higher the *NG* classification rates of **Find Border Max** and **Find Border Min**, and the lower their mean distances related to the exploration of the original connected components. This information could be expected, since these two algorithms can only suggest scenarios within the boundaries of their initial set. On the contrary, **Find Border Points** is able to perform well with good qualitative results whatever the initial set, as it looks everywhere in the initial space and is not restricted by the initial conditions.

This Section will end with results conducted with a different stopping criterion, pushing the algorithms up to 3,000 simulations, in order to check whether this changes the global trends of the proposed algorithms.

5.3.4 Quantitative Results – 3 000 simulations

We present here the results of a limited series of experiments for a stopping condition of 3,000 simulations. Only initial sets of 500 *NG* scenarios will be used for all three algorithms, since we have seen so far that the **Find Border Max** and **Find Border Min** algorithms perform best with a large initial set, and the **Find Border Points** delivers good results whatever the initial set size. Plus, we have seen that all three algorithms are able to minimize the border offset in all cases, but differ in the way they explore all the original connected components. Furthermore, we will only compare, on the one hand, the time each algorithm took to complete 3,000 simulations, and on the other hand, the evolution of the *NG* classification rate, as well as the coverage distances between the undetected *NG* scenarios in the original connected components and the real *NG* scenarios in the updated connected components. We will then be able to enforce our conclusions from the previous sections by checking whether the results of the algorithms improve if we allow a larger simulation budget, and if the **Find Border Points** algorithm remains the best algorithm in qualitative results.

Computational Cost

Figure 5.26 shows the time consumed by each algorithm in order to complete 3,000 simulations starting from an initial set containing 500 *NG* scenarios.

These results confirm that both **Find Border Max** and **Find Border Min** show a super-linear time complexity, respectively 15 hours for **Find Border Max** and 70 hours for **Find Border Min**, whereas **Find Border Points** displays a seemingly-linear complexity, requiring 30 hours. Hence, even though **Find Border Max** remains the fastest algorithm here, it is likely to become slower than **Find Border Points** if we increase even more the simulation budget. Taking into account the results for 1,000 simulations in Figures 5.2

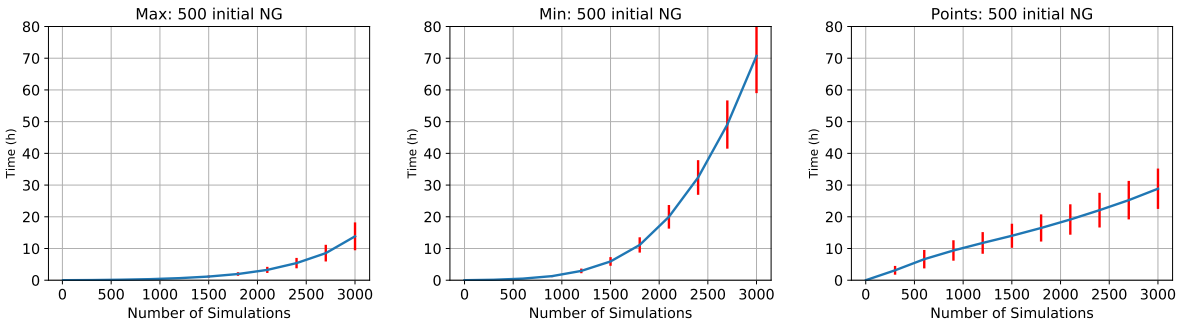

Figure 5.26: Average (and standard deviations) of computing times (hours) needed by **Find Border Max**, **Find Border Min** and **Find Border Points** to perform 3,000 simulations for initial sets containing 500 *NG* scenarios.

and 5.8, the number of simulations tripled, and hence the time needed by **Find Border Points** also tripled, whereas it increased by a factor of more than 40 for **Find Border Max**. Therefore, **Find Border Points** seems to be the most consistent from a computational complexity point of view. On the other hand, **Find Border Min** will clearly always be the slowest algorithm.

Next, we will evaluate the quality of the results of these experiments in a similar way than what has been done in Section 5.3.2.

5.3.5 Qualitative Results – 3 000 simulations

We will only look at the two qualitative indicators for which clear differences could be observed between the different algorithms in the 1000-simulations context: the *NG* classification rate, and the coverage distances.

Figure 5.27 illustrates the *NG* classification rate related to the connected components when comparing the results obtained by each algorithm to the grid. The nearest neighbor was considered when updating the grid evaluations. All algorithms run until 3,000 simulations have been performed, starting from an initial set containing 500 *NG* scenarios.

We notice that **Find Border Max** shows a slight improvement past the 1,000 simulations mark, as the mediocre rates for the lateral lane decentering distance criterion go up from 45% to 50%. This suggests that this algorithm would need even more simulations to be able to improve its rates considerably. On the contrary, **Find Border Min** is still unable to improve the lateral lane decentering distance criterion rates, which witnessed a dive in its values past the 1,000 simulations mark from 45% to 40%. However, **Find Border Points** manages to continue to boost its rates across all criteria, and especially for the lateral lane decentering distance criterion, hitting a final average value higher than 90%.

Besides, we recall that there seems to be a correlation between the *NG* classification rates and the coverage distances between the undetected *NG* original components scenarios and the real *NG* updated components scenarios. Therefore, Figure 5.28 displays the average

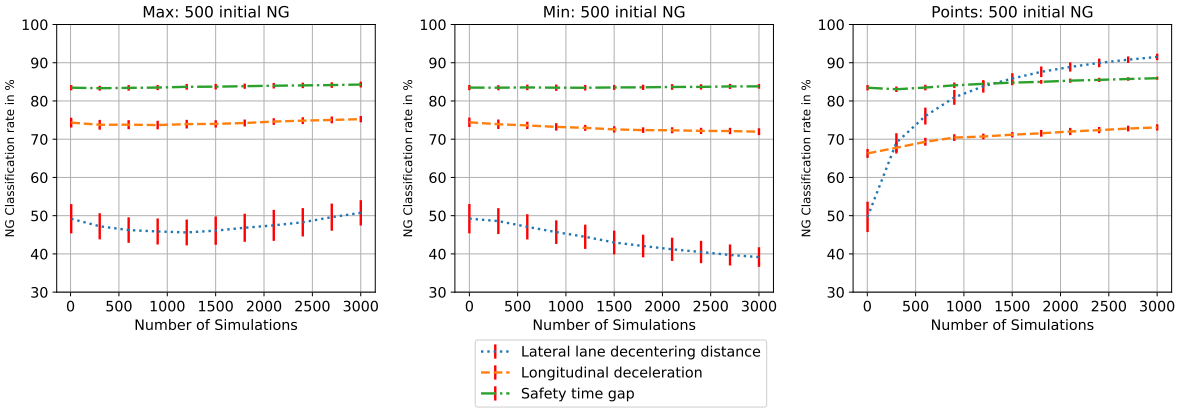

Figure 5.27: Means (and standard deviations) of the *NG* classification rate related to the **connected components** for **Find Border Max**, **Find Border Min** and **Find Border Points** running for 3,000 simulations.

values of these distances obtained by each one of all three algorithms during the completion of 3,000 simulations, starting from the initial set containing 500 *NG* scenarios.

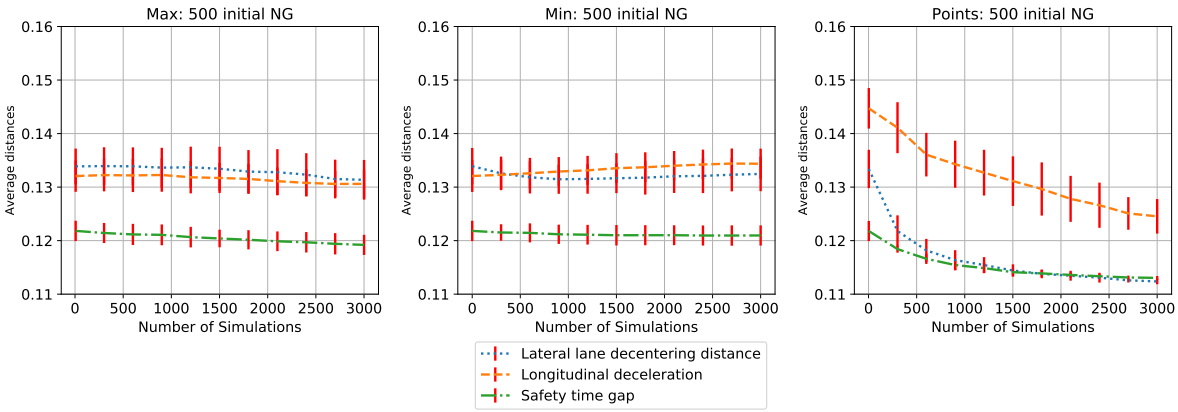

Figure 5.28: Mean *coverage distances* (and standard deviations) between the undetected *NG* scenarios of the original grid connected components and the real *NG* scenarios of the updated grid connected components after applying **Find Border Max**, **Find Border Min** and **Find Border Points** within a budget of 3,000 simulations, initial sets containing 500 *NG* scenarios.

First, we can see in this figure that **Find Border Max** shows slight improvements for all criteria beyond the first 1,000 simulations. The lateral lane decentering distance, in particular, begins to decrease from 0.135 to attain a final value of 0.13 at 3,000 simulations. Then, we can also observe that **Find Border Min** was unable to improve these distances results when extending the stopping condition limit, as all criteria display constant curves at best. Finally, **Find Border Points** continues at improving these distances to approach

the minimal precision distance $d_{min} = 0.11$ as much as possible. Notably, the longitudinal deceleration criterion, which is a challenge for this algorithm because of its discrete nature, displays a clear decrease in its distances results going down from 0.145 initially to a final value of 0.125.

Thus, there is indeed a correlation between these two qualitative factors. Although all algorithms manage to reduce the offset error in the border detection, not all of them are able to explore all the *NG* areas effectively throughout the simulations. **Find Border Points** is the most efficient algorithm qualitatively, **Find Border Max** arrives logically next since we saw a possible improvement debut at 3,000 simulations, and **Find Border Min** is the least efficient one when taking into account the time consumption and the qualitative results obtained. Nevertheless, the number of G/NG couples generated by **Find Border Min** is the highest amongst all three scenarios. Hence, each algorithm seems to operate in a different fashion, as they target the input search space differently.

Plus, these results are related to a single use case. Further tests should be conducted in order to see the efficiency of these algorithms in handling other autonomous vehicle situations. In particular, these algorithms will all be combined into the ADValue project at Renault in order to aid one another in the assimilation of the input search space for other use cases, while competing for the resources to generate scenarios. Besides, we combined the scenarios generated by all three algorithms starting from the initial set of 500 *NG* scenarios, and computed the resulting qualitative results shown in Figure 5.29.

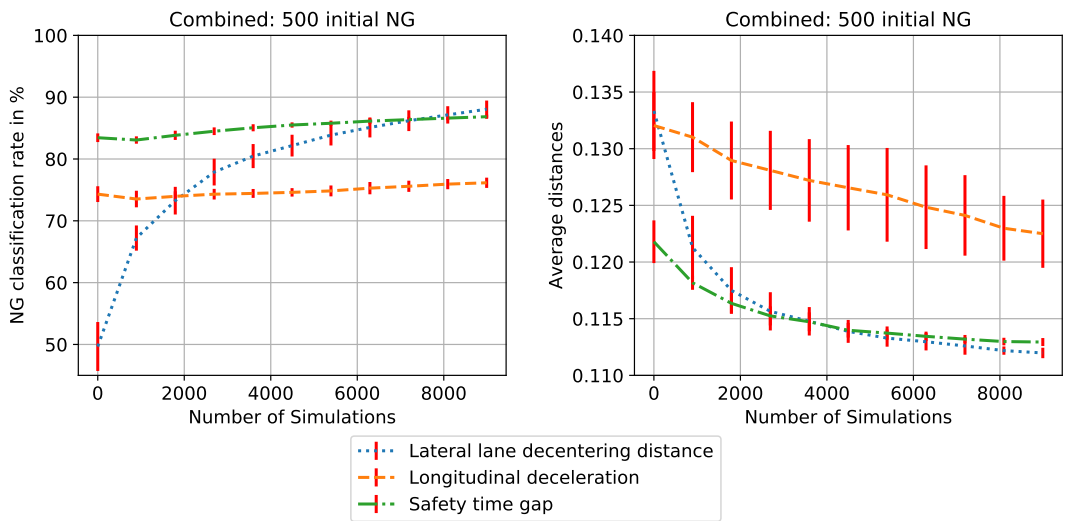

Figure 5.29: Variation of the *NG* classification rates and mean *coverage distances* after combining the scenarios generated by all **three** algorithms when attempting to reach 3,000 simulations with vertical error bars equal to twice the standard deviation values. Each curve is the result of 11 runs stemming from different initial sets containing 500 *NG* scenarios.

As expected, the combination of the algorithms shows acceptable qualitative results for

the NG classification rates and the coverage distances metrics. We also notice that the curves follow the trends displayed by **Find Border Points** which was the most efficient algorithm between all three. Thus, the idea of combining the algorithms will not affect negatively the overall results, but will, at best, follow the tracks of the most efficient algorithm. This further validates the notion of combining the algorithms to mix their efforts in exploring the input space instead of relying on a single algorithm to detect scenarios around the G/NG border.

5.3.6 Partial conclusion

This chapter has introduced and experimentally compared three algorithms designed to detect scenarios around the border between the G and NG areas of the use case input space. Starting with an archive containing a prescribed number of NG scenarios, all three algorithms repeat the following loop until their simulation budget comes to an end.

1. **Find Border Max** retrieves the G/NG pairs that are farthest apart in the scenario space, and proposes new scenarios between them, at distance d_{min}, the prescribed precision. Every G/NG pair (according to the reduced model) of scenarios very close to each other is a candidate border pair, that is checked with the simulator (substitution model in this work).

2. **Find Border Min** calculates the minimum distance between G/NG scenarios, then launches a dichotomy to retrieve the closest G/NG pair according to minimal precision distance d_{min}. Once more, this pair is expected to show a part of the border by passing between the scenarios of this G/NG couple.

3. **Find Border Points** operates on an optimization on the criteria directly, by identifying scenarios very close to the border, while respecting the minimal precision distance d_{min}. It launches CMA-ES with a small initial step size $= 0.01$, and tries to minimize the objective function which calculates the gap between the output criterion and its corresponding threshold.

Similar to the NG detection algorithm shown in Section 4.5, the three algorithms use the reduced model to comply to the industrial restriction of minimizing simulation costs, and the neural network to overcome software limitations by substituting temporarily the actual simulator. We assessed the performances of the algorithms by imposing a shared stopping condition of reaching a fixed amount of simulations, which is also in concordance with the industrial requirements. The comparison evaluation was conducted at three different factors: the time each algorithm consumed to achieve this stopping condition, the number of scenarios that were able to meet the given objective, and the quality of the results when compared to a grid of the same search space.

The **Find Border Max** and **Find Border Min** algorithms are characterized by super-linear running time and number of couples generated across the simulations, while the **Find Border Points** algorithm evolves in a more linear fashion. Although **Find Border Max** is

the fastest algorithm for 1,000 and 3,000 simulations, **Find Border Points** is deemed the most reliable one for a large number of simulations. Additionally, evaluating the qualitative results turned out to be a conceptual challenge. Precision rate and border rate metrics were considered, before finally settling on the connected components to assess the qualitative performances when comparing to a full grid over the input search space. This metric is flexible in detecting a possible offset in the border detection by each algorithm. The results show that all three algorithms are able to minimize this offset while approaching the maximum value, which is the minimal precision distance d_{min} that each algorithms respected while operating.

Nevertheless, two other qualitative indicators show the capacity of the algorithms in exploring effectively all the NG areas in the search space. **Find Border Points** is the best algorithm in managing to consistently improve throughout the simulations and effectively explore the NG areas related to all three criteria. The other two algorithms display slightly more mediocre results depending on the criterion evaluated, although **Find Border Min** manages to generate the biggest number of G/NG couples amongst all three algorithms, which indicates its ability to precisely detect the pairs in a particular zone. Plus, the intrinsic conception of the **Find Border Max** and **Find Border Min** algorithms limits their capacity in suggesting new couples outside their initial set, which explains the improvement of their qualitative results when enlarging the initial set. On the contrary, **Find Border Points** is able to propose new scenarios independently of its initial set and explore effectively the input space. And even when the algorithms results are combined, which will be the case in the ADValue project, the overall quality follows the tracks of the most efficient algorithm, and validates the idea of mixing multiple algorithms instead of relying solely on a single algorithm.

Equipped with these three algorithms, we are able to generate scenarios that are close to the G/NG borders of the input search space, whether as G/NG pairs, or "*on*"-the-border scenarios. These scenarios thus give an idea of an approximate localization of the border. Nevertheless, the three stochastic algorithms offer no guarantee as to the exact location of the border across the space of the input parameters. Although it effectively passes between the G/NG pairs or close to the "*on*"-the-border scenarios, the border can have various forms, and the number of scenarios to simulate to define it with more precision can increase very quickly. Therefore, alternative approaches are considered in the next chapter to direct our research towards a more in-depth study on border detection, based on border and failed scenarios which can be generated from all algorithms presented thus far.

Chapter 6

Border Models

This chapter addresses the third and final objective of the thesis: obtain an analytical or parameterized model of the border between the faulty and the non-faulty regions that is explainable. The rationale is that designing algorithms dedicated to either the direct identification of scenarios that result in a failure of the command law, or the identification of scenarios that are very close to this G/NG border, as done in the previous chapters, is not yet what the industrial experts want: they wish to be able to understand the root causes of the identified failures, the ranges of input parameters that result in failures, and not simply to correct their command law in a blind manner. Explainable AI has become a very important issue in current research (see e.g., the XAI research program launched by DARPA in 2017). Explainability is one component of trustworthiness, crucial for the societal adoption of AI in the society. But here another side of explainability is concerned, which is to allow the humans (here, the domain experts) to extract knowledge from the learned models, about the car command law, and allow faster further progress than by simply correcting the points of failure one by one. In particular, from the analytical formulation of the border, it should be possible to do some informed active learning, and propose the simulation of the most informative scenarios w.r.t. improving the global predictive accuracy of the model, while still complying to the industrial constraint of minimizing the number of simulations.

Building on the results of the border algorithms, and after having identified points on the border (or almost on the border), we are thus facing another learning problem that amounts here to symbolic learning (build a symbolic model of the border) more than to statistical learning. Two approaches have been tried, namely the exact solutions given by **Mixed-Integer Linear Programming (MILP)** (Section 6.3), and some approximate solutions found by **Genetic Programming** for Symbolic Regression (Section 6.4). However, and though not fulfilling the industrial specifications of explainability, **Neural Networks**, that have already been used in this work to derive the substitution model in the previous Chapters (Section 4.3.1), will also be used as some baseline (Section 6.2) that will set an upper bound on accuracy, as it is unlikely that the other methods can reach similar accuracies while preserving explainability: there is no free lunch in explainability.

Final preliminary words: all the experiments presented in the previous Chapters have been done on the tracking vehicle use case (Section 4.5.1), and many SCANeR simulations had been done for that use case. However, SCANeR itself had made considerable progresses during the course of this thesis, and its robustness had improved a lot: All the results used in Chapters 4 and 5 were considered suspect at the time of the experiments to find border models. Furthermore, the use case itself was considered far too simple to be convincing. Hence the results obtained up to now will not be used in this Chapter, and a new use case will replace the previous one, that we will describe in next Section.

6.1 The NHTSA 13 Use Case

The chosen use case consists of the autonomous vehicle (EGO) following another vehicle while approaching a lead vehicle moving at lower constant speed as seen in Figure 6.1.

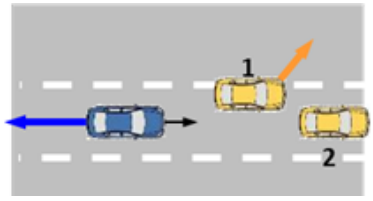

Figure 6.1: Use case NHTSA 13: Following Vehicle approaching lead vehicle moving at lower constant speed.

At the beginning of the simulation, EGO, Vehicle 1 and Vehicle 2 are on the same lane. Vehicle 2 is far ahead of the other two vehicles and stops moving, while EGO and Vehicle 1 are advancing with a velocity between 60 and 130 km/h. Then, when the Time To Collision (TTC) parameter between Vehicle 1 and Vehicle 2 passes below the TTC threshold fixed parameter, Vehicle 1 performs a lane change that lasts for a fixed duration of seconds. Meanwhile, EGO sees Vehicle 2 and should be able to brake to avoid collision. If the TTC between EGO and Vehicle 2 is lower (respectively higher) than 0.8 second anytime during the simulation, the scenario is labeled NG, (resp. G). The use case is of dimension 6, and all six input variables of this use case are described in Table 6.1.

This use case will be used to test the three algorithms that will be used to find border models, and compare their performances. Thanks to the evolution of the simulation software platform SCANeR along with the progress of the thesis, a complete run of a grid of 470,587 scenarios was pre-run for this use case, and will serve as basis for the evaluation process (as in Section 4.5 for the previous use case). The step sizes used for each input parameter are shown in Table 6.1. Note that when multiplying all possible levels, we obtain 534,600 possible scenarios. Then, a constraint of having the lateral shift duration of Vehicle 1 less than or equal to the TTC threshold parameter is applied to eliminate the scenarios where Vehicle 1 crashes into Vehicle 2 while performing the lane change. Out of the 475,200 scenarios left,

Table 6.1: Description of the input parameters of the use case NHTSA 13.

Input parameters specifications				
Name	**Unit**	**Range/List**	**Step size**	**Levels**
Initial velocity of EGO	km/h	$[60, 130]$	5	15
Initial velocity of V1	km/h	$[60, 130]$	5	15
Initial time between EGO and V1*	s	$[1.5, 2.5]$	0.1	11
Lateral shift duration of V1*	s	$[0.7, 1.2]$	0.25	3
TTC threshold between V1 and V2*	s	$[0.7, 3]$	0.1	24
Type of V2*	-	Car, motorcycle, bus	-	3

*V1 and V2 refer to Vehicle 1 and Vehicle 2 respectively in the use case.

470,587 scenarios were successfully simulated with SCANeR Studio.

Although this simple use case has a small dimension, it is nonetheless considered to be realistic as far as the simulation allows it. For instance, adding these last constraints on the use case eliminates irrelevant scenarios that cannot occur physically, and by that enhances its degree of realism. Obviously, more options could be added to the simulation to represent more accurately the scenarios as compared to the real world (e.g. weather, slippery road, other traffic vehicles...). The use case definition is realized by a specialized simulation team at Renault while staying in concordance with the real capacity of the SCANeR simulator itself in being the most realistic possible. Knowing that the SCANeR software is in constant evolution, the realism of the simulated use cases should gradually improve.

Additionally, for each scenario, the status (G or NG) of all its neighbors (Cartesian coordinates plus diagonal neighbors) is recorded, and if a scenario has both G and NG neighbors, it is considered to be "on" the border. Thus, we now have a complete set of G/NG scenarios with additional information of whether they are "on" the border (140,789 are considered on the border, out of the 470,587 scenarios) or not (the remaining 329,798 scenarios). This will be considered as the ground truth when building and developing the different border model algorithms. An alternative would have been to run the three algorithms from Chapter 5, **Find Border Max**, **Find Border Min**, and **Find Border Points**. However, this is left for further work, as we wanted to see how efficient and useful the border models could be in the best possible setting (known ground truth for the border scenarios).

6.2 Neural Network Border Models

Neural Networks have been introduced in Section 4.3, and used to build the substitution model (Section 4.3.1) that was used in lieu of the simulator in Chapters 4 and 5. Their predictive accuracy is well known, and has been demonstrated in many domains, as discussed previously. However, such performances can only be obtained when they are trained on huge datasets, and the industrial specifications of this work include minimizing the number of simulations. But more importantly, and this is the main motivation of deriving border

models, the explainability of the results is crucial for domain experts. And this is where Neural Network fail. Indirect explainability (i.e., by setting up specific mechanisms together with the Neural Networks to explain them) could be a solution, but this is not the path that was chosen in this work, and it would somehow contradict the idea of deriving border models.

Hence the motivations to try Neural Networks as a border model are twofold: First, obtain some bound for the accuracy of possible border models, assuming Neural Networks are the best approach to learn an accurate model here, provided enough training examples are available; Second, see how the performance of Neural Networks degrades as the size of the training set decreases, and how other models could fill this gap. As a consequence, however, only few trials have been performed with Neural Networks, and no systematic experimental campaign including several runs per setting and statistical analysis of the result has been run, leaving the results as mere indications about the behavior of NN models, and saving time for the a priori more promising MILP and Genetic Programming models that will be developed in Sections 6.3 and 6.4 respectively.

6.2.1 Methodology

The goal of these experiments is to build a Neural Network that identifies the border of both G and NG areas. Because the ground truth is available (see above), we will directly learn if a scenario lies on the boundary or not: this is a classification problem, and standard setting is chosen after a few initial experiments to tune the hyper-parameters. The only parameter that we will experiment with is the size of the training set, varying from 10,000, 5,000, 2,500 to 1,000. The **accuracy** of the resulting models will be measured on the whole set of 470,587 scenarios, for which the ground truth is known. Furthermore, in an attempt to improve the global performance of the method, an active learning iterative procedure will be proposed and experimented with. These different components will now be detailed.

The Neural Network The Neural Network has one input layer with 5 continuous input variables and one hot-encoded 3-values variable, three hidden layers with 150, 100 and 50 neurons and ReLU activation functions, and one output layer with sigmoid activation function for this Boolean classification task. The cost function is the cross-entropy (Equation 3.5). The learning rate has been set to 0.001 while using the Adam optimization algorithm. The Neural Network is trained for 1,000 epochs with a batch size of 64 scenarios.

The Datasets Different dataset sizes have been used. In all cases, the original dataset is made of equal numbers of "on"-the-border and "off"-the-border scenarios, independently from their actual G/NG status. This sample set is then split into training and test sets according to a 67%-33% ratio. No validation set is used here, all hyper-parameters having been decided once and for all from some preliminary experiments.

The Iterative Active Learning Procedure We have implemented the following iterative procedure. Given a learned NN model, we compute its output on the whole grid of 470,587 scenarios, and order them by proximity of the NN output with 0.5 (whereas 0 means "off"-the-border and 1 "on-"the-border). The scenarios with output close to 0.5 are the ones for which the NN is the most uncertain of their class. A number of these uncertain scenarios (unless otherwise specified, we will randomly choose 50 scenarios whose predicted outputs are in [0.45,0.55]) is then added to the training set, and a new model is learned from scratch. The process can be repeated several times.

6.2.2 Results

First Results and Robustness Issue First experiments dealt with 10,000 scenarios (5,000 "on"-the-border and 5,000 "off"-the-border, as discussed). Note that in the framework of Deep Learning, this is a rather small dataset, while in the industrial context we are working in, this is already very large. Figure 6.2 shows the evolution of the cross-entropy loss along the learning epochs. In this case, the training and test accuracies are equal to 98.85% and 97.52% respectively.

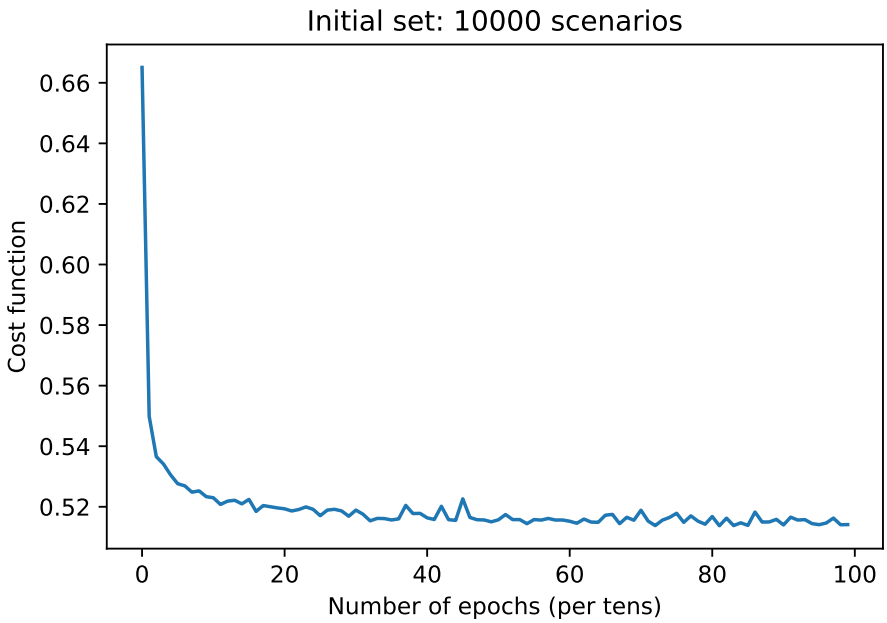

Figure 6.2: Cost function of the Neural Network border model during training phase for an initial set of 10,000 scenarios.

Using this model to predict the output values of all 470,587 scenarios of the full grid, we obtain a total error of 2.81%, where 1.87% correspond to the 140,789 border scenarios and 3.22% to the remaining "off"-the-border scenarios. The total error is very acceptable, and it looks that Neural Networks are reliable border models as far as accuracy of the prediction is concerned, though totally lacking explainability, as discussed.

Unfortunately, it seems that the NN model, in this situation, also suffers from a clear lack of robustness. When performing different independent runs on the same training set, even though the majority of the runs produced similar plots than the one of Figure 6.2, we commonly obtained results like the one displayed on Figure 6.3, with some abnormal peaks in the loss, and an asymptotic value of around 0.57 (compared to the approx. 0.51 of most runs), and corresponding final accuracy values of 90.21% (train) and 89.18% (test) for Figure 6.3 compared to the 98.85% and 97.52% for Figure 6.2.

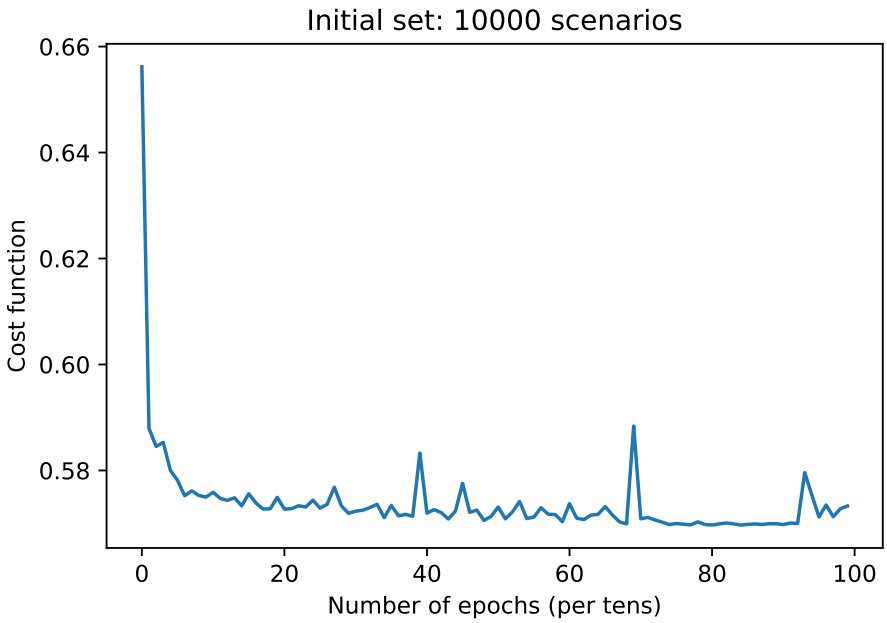

Figure 6.3: Evolution of the cost function of the Neural Network border model during training phase, while attaining local minima, for an initial set of 10,000 scenarios: 5,000 border scenarios and 5,000 other scenarios.

In order to tackle this issue, several options were explored: Increasing the number of epochs; Changing the optimizer from Adam to Stochastic Gradient Descent or RMSProp; Implementing some learning rate decay; Modifying the topology of the network (number of layers, number of neurons); Using regularisation techniques, e.g., dropout and L2 regularization. But neither one of these trials was successful in removing these instabilities. In the end, and because Neural Networks are anyway not the model of choice because of explainability reasons, we stopped trying to robustify their results, and will simply report a few results addressing the iterative active learning procedure, and the influence of the dataset size when training the network on small datasets.

Results of the Iterative Active Learning Procedure Figure 6.4 illustrates the evolution of the training and test accuracies along 50 iterations of the active learning procedure. A first positive result is that all models achieve more than 95% in test accuracy. Unfortu-

nately, no improvement is to be seen as iterations proceed, even with a slight decrease of both training and test accuracies, and even though the dataset is augmented by 50 new scenarios at every iteration (hence 2,500 in total).

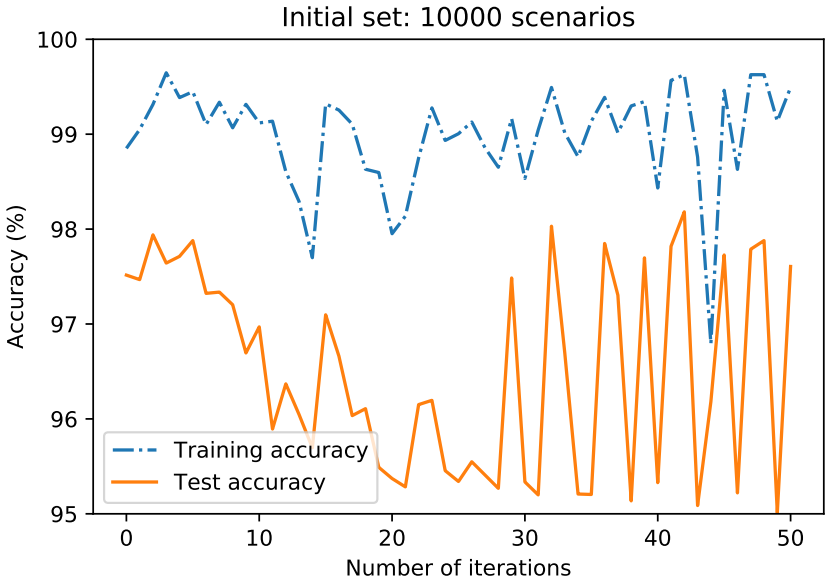

Figure 6.4: Variation of the training and test accuracy of the Neural Network throughout the methodology iterations, trained on an initial set of 10,000 scenarios: 5,000 border scenarios and 5,000 other scenarios.

And this tendency is confirmed by Figure 6.5, that shows the variations of the errors computed between the Neural Network predictions and the real labels of the 470,587 scenarios (the 140,789 border scenarios and the 329,798 non-border scenarios): the total error is globally unchanged along the iterations, with slight oscillations around the initial values, and again an instability at iteration 44 with a 4% error (almost 6% on the non-border scenarios).

Influence of Dataset Size The previous results were obtained using datasets of size 10,000. As discussed, though rather small in the Deep Learning context, this size is already rather large in the industrial context of this work. We will consider in this Section datasets of sizes 5,000, 2,500 and 1,000, containing half on-the-border and off-the-border scenarios.

The same iterative active learning procedure was applied, and the results can be seen in Figure 6.6: for each dataset size, we display the train and test accuracies along the active learning iterations, and the global errors on the whole grid of 470,587 scenarios. The general trend is that both performances decrease when the dataset size decreases, something that could be expected. For a size of 1,000 (the smallest we tried), the test accuracy even fell below the 5% threshold deemed acceptable in the industrial context we are working in. Furthermore, the lack of robustness of the whole process clearly appears, with unexpected peaks (both positive and negative) appearing in all figures.

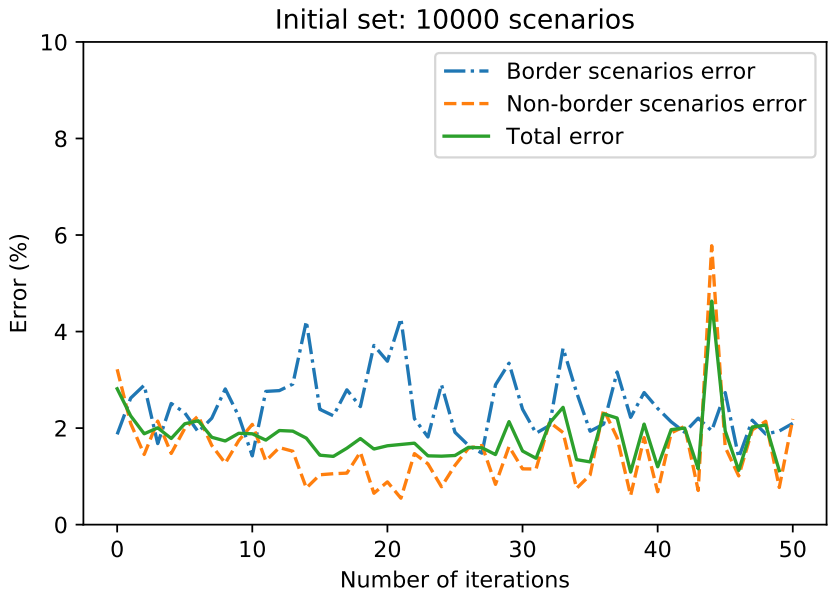

Figure 6.5: Variation of the error predicted on the border and non-border scenarios of the 470,587 scenarios of the full grid throughout the methodology iterations, adding up to the total error, using a Neural Network trained on an initial set of 10,000 scenarios: 5,000 border scenarios and 5,000 other scenarios.

6.2.3 Conclusions regarding Neural Network as Border Models

The conclusions drawn from these limited experiments should be considered cautiously, as they are not backed up with statistically significant results. But, as explained in the introduction of this Section 6.2, Neural Networks were a priori not a good candidate for border models, for their lack of explainability. Considering the wide application areas of NNs and their overall outstanding performances in many domains, these results can nevertheless set a lower bound on the accuracy one can expect when the ground truth regarding the border between G/NG is available, and we shouldn't expect getting much better error rates than the 2.81% obtained here with a dataset of 10,000 scenarios, whatever the method we try next.

Further than that, these results confirm the importance of the dataset size for Neural Networks models. And if Neural Networks are to be considered as an efficient border model, a trade-off must be made between the dataset size and the error between the Neural Networks border predictions and the real labels. Still, it might be considered to be an acceptable border model across the 10,000, 5,000 and 2,500 experiments, reaching the 95% accuracy threshold, and if this acceptable threshold is increased to 10%, the 1,000 experiment also leads to an acceptable model. In any case, they offer a baseline for comparison with other scientific approaches.

But it should be remembered that, up to here, an important drawback of Neural Networks used as border model is this lack of robustness we were unable to understand and fight. But

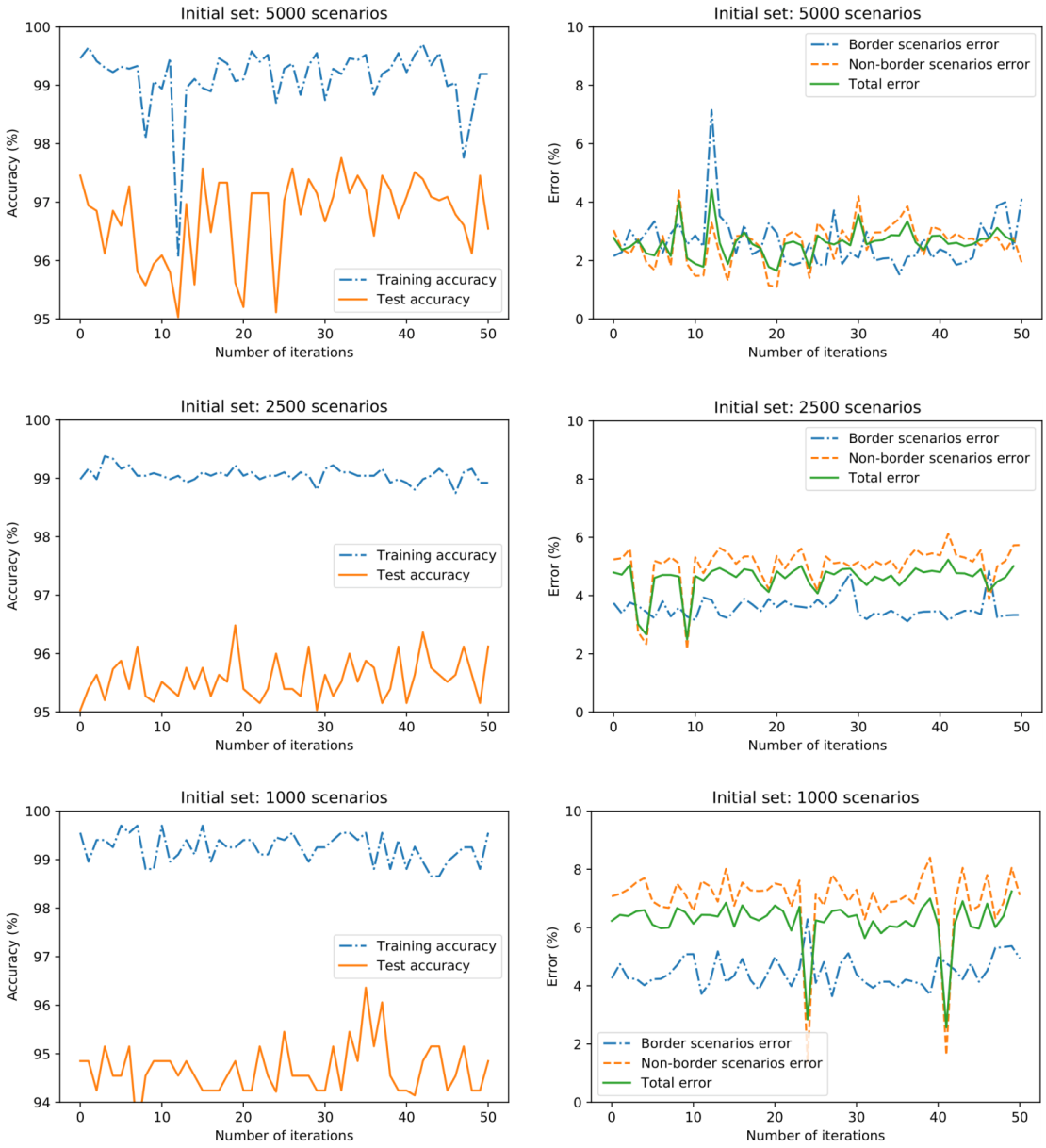

Figure 6.6: Training and test accuracy of the Neural Network (left column), and error predicted on the border and non-border scenarios of the 470,587 scenarios of the full grid (right column) throughout the methodology iterations, trained on initial sets of 5,000, 2,500 and 1,000 containing equally border and other scenarios.

the main drawback in the context of this thesis is that no easily interpretable equation can be derived from NN border models, resulting in a total absence of explainability of the resulting model, something the domain experts think is critical. The approach presented next, based

on mathematical programming, is considered for that goal.

6.3 MILP based Border Models

6.3.1 Introduction

Linear Programming (LP) is a mathematical optimization program which attempts to maximize or minimize a linear objective function, on which are imposed one or more linear constraints. An Integer Program (IP) is an optimization model in which variables are restricted to be integers, and a Mixed-Integer Linear Programming (MILP) is a linear program in which some variables are restricted to be integers, whereas others are continuous.

The rationale for using the MILP formalism is that there exist several solvers to provably exactly solve MILP problems – though sometimes at very high (or even too high) computational costs. As exact solvers, MILP solvers lack the flexibility of stochastic solvers like Neural Networks or Genetic Programming. However, when the combinatorial complexity becomes too high, MILP problems can be relaxed by introducing ad hoc slack variables that allow the user to tune the amount of error s.he is willing to accept.

In this Chapter, we assume that we have a dataset of N scenarios for a given use case, that have been labeled G or NG according to a given command law under validation, and we want to create a MILP model that can separate the G scenarios from the NG scenarios. The result of the MILP solution will be mathematical equations that represent the border located between the G and the NG regions of the input space, as defined by the available samples. By choosing a MILP model, we aim to generate border models that are explainable (the equations) and separate the G/NG scenarios with a controlable error rate (in particular being able to derive exact models, computational cost permitting) - two features that stochastic approaches like Neural Networks lack.

We will proceed in several steps and derive more and more complex MILP problems, and the associated optimization procedure, to be able to handle a real-world use case such as the use case NHTSA 13 (see Figure 6.1). For the first step, we will consider the simple case where the NG zone is made of one single connected area (Figure 6.7-left), and derive one single MILP model that will be solved directly.

6.3.2 Global MILP Model

Our first approach to derive mathematical equations for an exact representation of the border according to a set of G/NG scenarios formulates this problem into a single MILP.

Problem setting An example of the desired model output is shown in Figure 6.7. The left plot shows a two-dimensional input space (e.g., consisting of the acceleration and deceleration of the EGO vehicle), with G and NG scenarios represented as green and red circles respectively, as well as the target border, the straight lines that, together, effectively separate

both categories. Because we assume that the NG zone is one single connected area, we can represent such border in a tree-like structure, fixed beforehand, in which in particular the number of lines (branches of the tree) is set in advance. The tree corresponding to the left hand side example of Figure 6.7 is represented on the right-hand side. Each line, or axis, delimits solely G scenarios on its left side, and on its right side, the rest of the scenarios, that can be G or NG. The ultimate goal is to keep on creating as much axes as needed to ultimately capture all the NG scenarios only at the right side of the final axis. In other words, a scenario will be considered G if it had been classified as G by one axis, whereas a scenario is evaluated as NG if no axis classifies it as G.

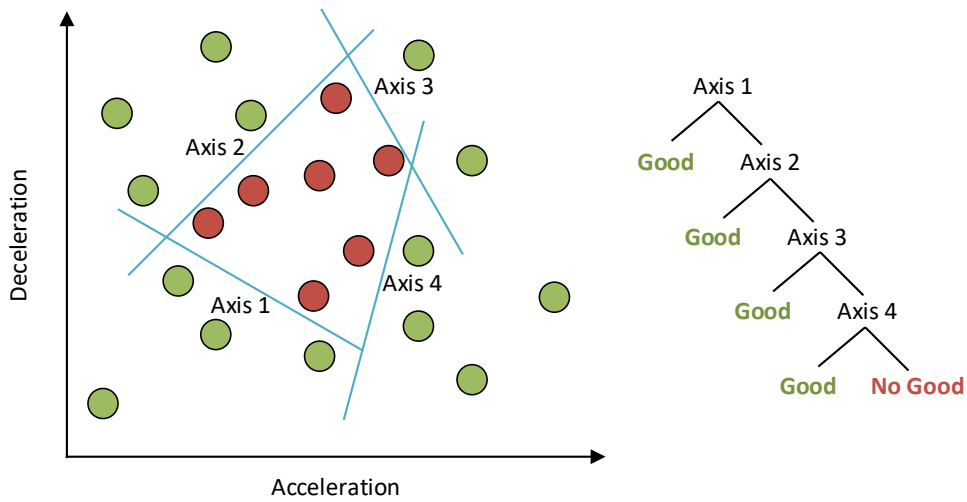

Figure 6.7: 2-D representation (left) of the desired MILP model which finds the axes that separate NG red scenarios from G green scenarios in a tree-like structure (right).

This representation can be cast into the following equations:

$$\forall i \in [1, N_G], \exists k \in [1, K], \quad \sum_{j=0}^{n} \alpha_{kj}.X_{ij} \leq 0 \tag{6.1}$$

$$\forall i \in [N_G + 1, N], \forall k \in [1, K], \quad \sum_{j=0}^{n} \alpha_{kj}.X_{ij} \geq 0 \tag{6.2}$$

where N is the number of examples, assuming the first N_G examples are G and the others NG, n is the dimension of the scenario space, K is the number of axes. X_{ij} is the j-th input value for the i-th scenario, with the convention that $X_{i0} = 1$, allowing a constant term in the equation of axis k, and α_{kj} are the unknowns, $(\alpha_{kj})_{j\in[0,n]}$ being the coefficients of the linear equation defining the k-th axis (in fact, a hyperplane, but referred to as "axis" in the following) in the input space. These equations simply represent the projections of the

scenarios on the axes, with a sign convention chosen as negative for the G scenarios and positive for the NG scenarios.

Even without being yet a MILP formulation, Equations (6.1) and (6.2) involve $N \times K$ constraints, and N will typically range from a few hundreds to several thousands. Furthermore, as such, on the one hand there exists probably an infinite number of solutions to these two equations, and on the other hand, all unknowns α set to zero is a trivial solution. Hence additional conditions are needed to make the problem well-posed.

The MILP problem The problem set up by Equations (6.1) and (6.2) can classically be posed as a MILP by first introducing Boolean variables to handle the "$\exists$" condition, and by duplicating the unknowns α_{kj} to take care of the sign conditions, the latter in turn leading to some objective function to minimize in order to avoid an infinite number of solutions.

The constraints of Equations (6.1) and (6.2) are hence rewritten as:

$$\forall i \in [1, N_G], \forall k \in [1, K], \quad \sum_j (\alpha_{kj}^+ - \alpha_{kj}^-).X_{ij} - (1 - B_{ki}).BIG \leq 0 \tag{6.3}$$

$$\forall i \in [N_G + 1, N], \forall k \in [1, K], \quad \sum_j (\alpha_{kj}^+ - \alpha_{kj}^-).X_{ij} + \cancel{(1 - B_{ki}).BIG} \geq \delta \tag{6.4}$$

Boolean variables B_{ki} are defined for each scenario i and axis k-th axis, by $B_{ki} = 1$ if scenario i is classified as G by axis k, and BIG is a large positive real constant that ensures that Equation (6.3) is true whenever B_{ki} is 0, and in particular for NG scenarios. Therefore, we will be able to deduce from the B_{ki} which axis actually classified scenario i as G. Furthermore, an additional constraint must be set to force the model into considering that one axis classifying each scenario as G is sufficient:

$$\forall i \quad \sum_k B_{ki} = 1 \tag{6.5}$$

Hence, for a given scenario i, only one Boolean variable corresponding to one axis can be set to 1 whereas all others related to the other axes have to be equal to 0. The same reasoning could be applied to the NG scenarios. However, since we have currently started by considering that there is only one NG zone, as represented for instance in the tree representation of Figure 6.7, then there is no need to create similar Boolean variables for the NG scenarios. Indeed, all axes should classify the NG scenarios as NG.

Another important change of variables was made between Equations (6.1) and (6.2) and Equations (6.3) and (6.4): the unknowns α_{kj} have been duplicated into $\alpha_{kj}^+ - \alpha_{kj}^-$, with $\alpha_{kj}^+ > 0$ and $\alpha_{kj}^- > 0$. There was no constraint on the sign of the α_{kj}, but for optimization purposes, it is more beneficial to deal with strictly positive variables. Hence each α_{kj} is now expressed as the difference of two strictly positive real variables. However, this introduced the possibility of infinitely many solutions, with identical values for $\alpha_{kj}^+ - \alpha_{kj}^-$ though different

values for both. Hence an additional objective function, to be minimized, is added, avoiding such trap:

$$\min \quad \sum_{k=1}^{K}\sum_{j=0}^{N}(\alpha_{kj}^{+} + \alpha_{kj}^{-}) \tag{6.6}$$

Finally, in order to forbid the trivial solution where all α's are set to zero, a small positive constant δ has been added to the right hand side of Equation (6.4). However, and we will see examples of such situations below, this opens the door to possible errors, and the value of δ should be carefully adjusted in a problem-dependent way. Unless otherwise specified, a default value of 0.01 was first used in all experiments in this thesis.

Results After defining the problem, the next step is its implementation in an existing MILP solver. Our experiments used **lp_solve**, an open-source MILP solver in Python (Berkelaar et al., 2004). Furthermore, before trying this model on the use case NHTSA 13, we have built several two-dimensional test cases of increasing difficulty for easier testing and validation of the proposed approach.

The simplest test case, test case #1, is described in Figure 6.8 : The dataset is made of 50 points, including a NG corner area that can be delimited by a single straight line. The right plot on Figure 6.8 shows that `lp_solve` managed to find quickly an accurate axis that separates the NG scenarios from the G scenarios. The key advantage of this method is that it finds the exact mathematical equation of the G/NG separation, i.e., we obtain the following equation for this first test case: $X2 = 1.01 - 0.178.X1$.

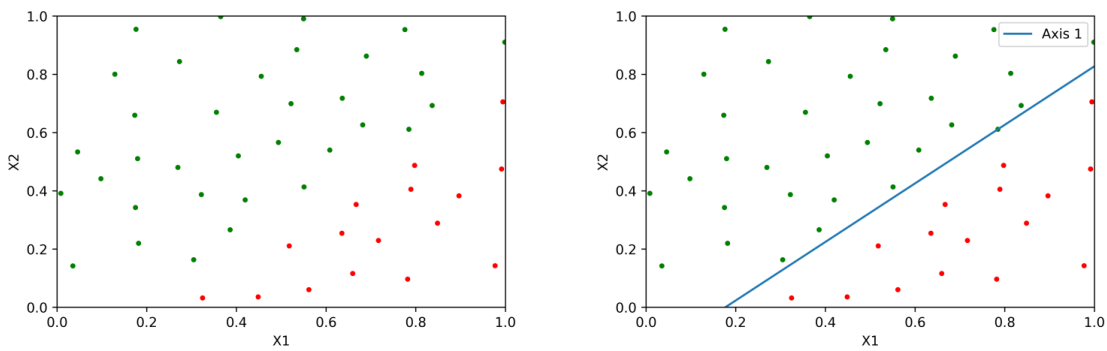

Figure 6.8: Test case #1: The 50 scenarios (left) including G scenarios (green) and a single NG corner zone (red), and the MILP model (right).

Test case #2.1 is slightly more complex: the NG area is moved in the middle of the input space. The NG zone consists of a disk of center (50,50) and a radius of 10. The results generated by the MILP solver can be seen in Figure 6.9.

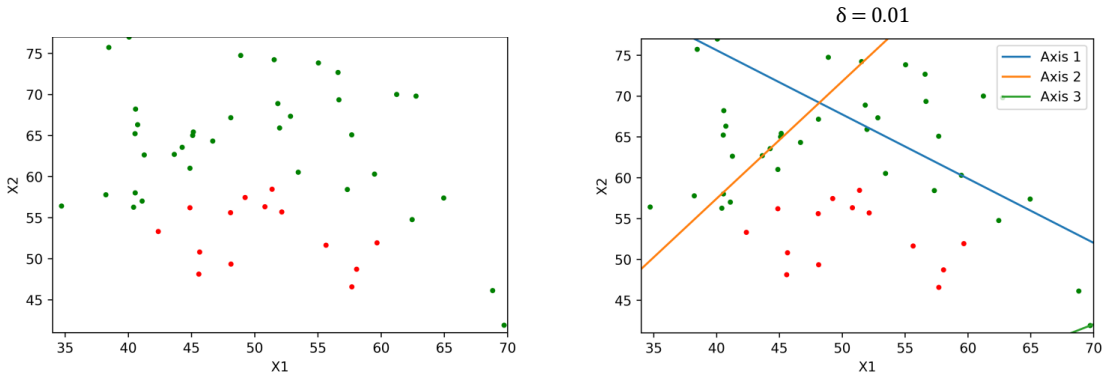

Figure 6.9: Test case #2.1: The 50 scenarios (left) including G scenarios (green) and a single NG disk zone (red), and the MILP model (right) with $\delta = 0.01$. Notice the third axis at the very bottom right.

Since this NG zone cannot be delimited by a single straight axis to separate it from the remaining G scenarios, the program crashes if we set the number of allowed axes to 1 like for test case number 1. Thus, the solver is only able to provide successful results for a number of axes set to 3. However, we notice that the axes are inaccurate in drawing exact G/NG scenarios, as we can easily notice G scenarios mixed with the NG scenarios on the wrong sides of the axes.

After effective testing, it turns out that the problem is an additional offset brought to the axes by the solver. This offset is directly linked to the δ parameter value found on the right side of the NG constraints equations. A small change in its value can greatly affect the solver output. We can compare this parameter to the hyper-parameters of the Neural Network, which should be carefully tuned to guarantee successful results. For instance, when setting δ to a value equal to 0.05 instead of 0.01, the solver is able to generate effective results just as shown in Figure 6.10. The three axes produced capture well the NG scenarios apart from the G scenarios.

Nonetheless, it is clear that if using straight axes, their required number can easily increase to encompass all NG scenarios, especially if the NG area has a sharp curvature. In such situations, the number of coefficient and Boolean variables will also increase accordingly, leading to a possibly huge complexity of the resulting MILP problem. In order to overcome this difficulty, and because we also want to be able to produce results that are accurate w.r.t. the real border, we decided to further enrich our model by allowing it to use conic surfaces instead of straight hyperplanes. In order to do so, we added as input variables to the MILP all second-degree monomials of the base input variables of the scenarios, i.e., the products of any two initial variables. This of course lead to adding new α coefficients. We applied this "quadratic" option to the test case #2.1. The results can be seen on Figure 6.11.

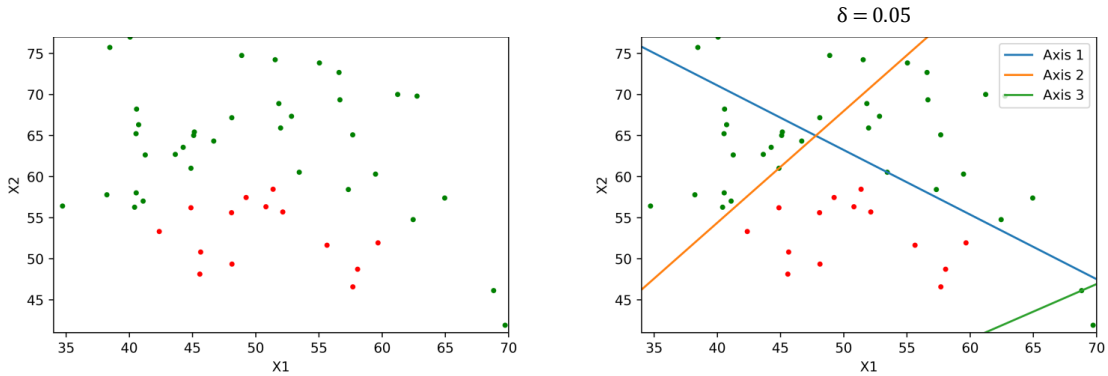

Figure 6.10: Test case #2.1: The 50 scenarios (left) including G scenarios (green) and a single NG disk zone (red), and the MILP model (right) with $\delta = 0.05$.

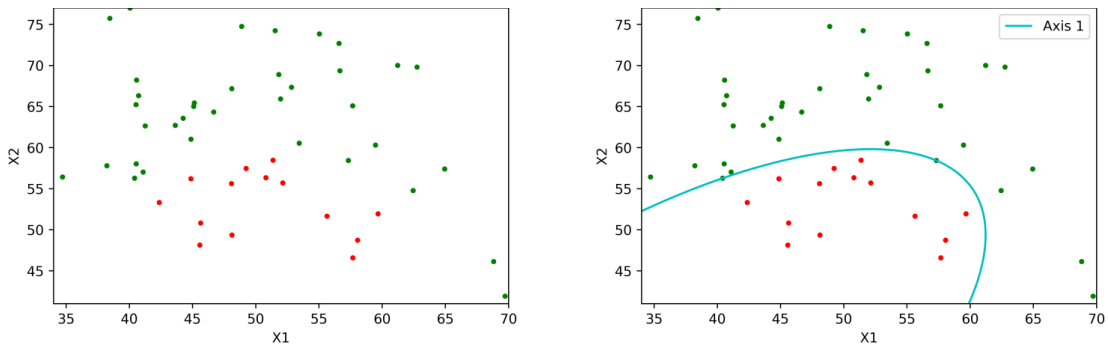

Figure 6.11: Test case #2.1: The 50 scenarios (left) including G scenarios (green) and a single NG disk zone (red), and the MILP model (right) using the quadratic option.

This quadratic option allowed `lp_solve` to find an axis that is a part of an ellipse, its equation including $X1.X2$, $X1^2$ and $X2^2$ terms. Thus, we have successfully reduced the number of axes by replacing three straight axes with a single elliptic curve that exactly delineates the NG scenarios for the same problem. Nonetheless, it is still far from the original circle shape of the NG border that we are trying to exactly identify. This is mainly due to the absence of points below the NG scenarios, making it impossible for the solver to figure out that the desired output is a circle.

In order to check this hypothesis, we sampled 200 scenarios of the same use case, instead of the 50 scenarios of test case #2.1. The updated test case #2.2 and the corresponding results are shown in Figure 6.12.

The solver clearly managed to find a much better approximation of the circular border when enough points were available in the input space. This newly discovered notion sheds

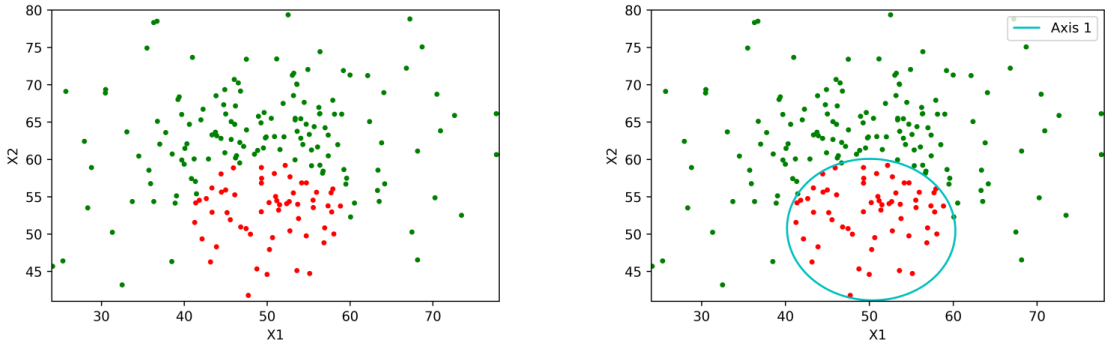

Figure 6.12: Test case #2.2: The 200 scenarios (left) including G scenarios (green) and a single NG disk zone (red), and the MILP model (right) using the quadratic option.

some light on the next step that should be implemented with this model; after the solver produces the outputs to the problem, the next interesting idea would be to find where to add new points, close to the axes generated, in areas that are not too populated by the current training set. After simulating them to get their true status (G or NG), we would launch the solver once again to check whether the newly added points modified the identified border. The process could be iterated until no significant change is detected.

A preliminary investigation of this approach was tested on a another slightly more difficult test case. Test case #3 features 200 points that contain a NG area located between two concentric circles. Outside the outer circle and inside the inner circle are the G areas. We want to evaluate whether the solver will be able to detect the two circular borders if we add some points close to the axes generated by the MILP model resulting from a first run of `lp_solve`. In order to find which points to add, we used here CMA-ES, with the objective function to be minimized being the axis equation itself: CMA-ES will try to find new points that minimize the axis equation, thus getting as close as possible to the part of the border represented by this axis. In a way, this is similar to the **Find Border Points** algorithm described in Section 5.2.1. To illustrate this approach, Figure 6.13 shows an example of its application on test case #3: the original test case, the axes found by a first application of `lp_solve`, and those found by a second application of `lp_solve` after the addition of 100 scenarios using CMA-ES.

First, the middle graph confirms the benefit of adding the quadratic option to this problems, which simply could not be resolved with only straight axes. The solver successfully managed to find two curves that delimit the NG area from the G scenarios in the search space. Although the solver managed to draw the outer circle border, it was unable to capture the inner circle border well and detected instead a rather elongated ellipse. This is mainly due to the absence of points in this particular area. CMA-ES is then launched 100 times, with objective function the equation of this inner axis (to be minimized). The right plot shows

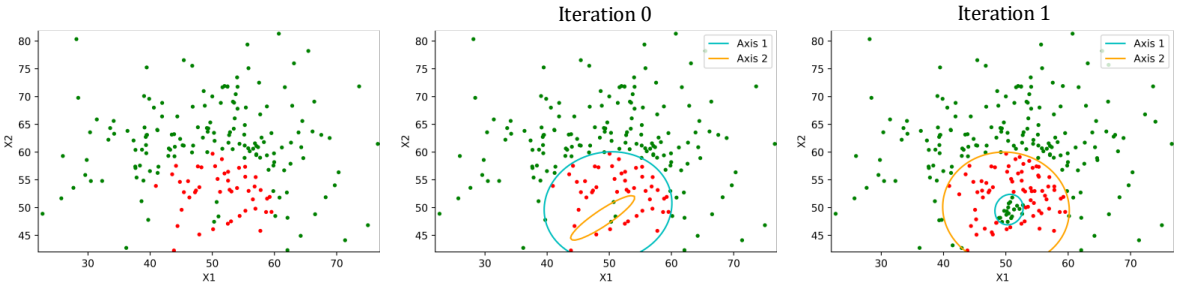

Figure 6.13: Test case #3: The 200 scenarios (left) including a single NG area (red) located between two concentric circles separating them from the remaining G (green); the first MILP model (center) with the quadratic option; the second MILP model after introducing 100 new points close to the axes of the first model found (right).

these additional points, and the new axes found by the MILP solver, that approximate the inner circle in a much more satisfactory manner. The thorough investigation of this **active learning** process is left for further work, in particular with the introduction of additional criteria to choose the new points to be simulated (e.g., using some of the "Find Border" algorithms of Chapter 5).

Results on NHTSA 13 The proposed approach showed promising results on the simple two-dimensional test cases with varying difficulty: it was time to test it on the use case NHTSA 13 (Section 6.1). It is a six-dimensional use case, there are hence 27 input variables with the quadratic option.

The first test used a small training set of 100 scenarios, containing equally 50 border and 50 off-border scenarios. Using the linear option, the solver is unable to find an appropriate solution in a reasonable amount of time even though the number of axes allowed had to be increased consecutively to 5, 10, 20 and up to 50. When using the quadratic option, the solver was able to instantly find a corresponding axis that did minimize the objective function. This can reflect the shape of a real use case border to be more curved than straight. We then calculated the error on the whole full grid of 470,587 scenarios of this MILP model, and obtained a global misclassification error of 7.7%, which shows that this methodology can be really effective.

The size of the training set was then increased to 200 scenarios, including equally 100 border and 100 off-border scenarios. The solver took around 5 minutes to end, and was still able to detect two axes with the quadratic option. The total error of that model was equal to 5.3%, logically better than with a smaller training set, and industrially acceptable. However, when trying to go for an even bigger training set with 500 scenarios, the solver was unable to solve the problem in a reasonable amount of time, due to the increased complexity reflected by the big number of variables. Industrial restrictions impose having a model that is able

to produce results in a matter of minutes, especially if it is to be used intensively for the validation of the autonomous vehicle through the simulation platform. Hence, two possible directions were considered: either test a more powerful commercial solver, able to deal with such higher complexity, or adapt the mathematical model and try to decrease its intrinsic complexity.

We began by testing a commercial solver, namely the **Gurobi** Optimizer, a fast commercial optimization solver for MILP problems (Gurobi Optimization, 2020), for which we were offered a free academic license for testing purposes. Furthermore, it was easy to redefine the problem within the **Gurobi** framework. After some preliminary test on the different 2-D toy problems introduced above, double-checked against `lp_solve`, it was run on NHTSA 13. Table 6.2 shows the successful tests applied on this real use case by considering different sizes for the training set, and showing the number of axes used, the overall time consumed by the solver, and the corresponding misclassification errors computed on the full grid of 470,587 scenarios. For each training set size, 11 independent runs were performed with 11 different training sets, and the Table 6.2 reports the mean and standard deviations of the results.

Table 6.2: Global MILP results using **Gurobi** solver on the NHTSA 13 use case.

Set size	**Linear / Quadratic**	**Number of axes**		**Time (sec)**		**Misclassification error (%)**	
		Mean	SD	Mean	SD	Mean	SD
100	Linear	4.0	0.0	0.39	0.11	10.64	2.63
100	Quadratic	1.0	0.0	0.15	0.02	7.38	1.73
200	Quadratic	2.0	0.0	2.54	1.33	7.31	1.54
500	Quadratic	2.0	0.0	63.63	79.76	4.2	0.76
1000	Quadratic	2.0	0.0	908.04	838.9	2.67	0.42

In contrast with `lp_solve`, the **Gurobi** optimizer is able to successfully solve the problem with the linear option based on the set containing 100 scenarios with equal distribution between G and NG scenarios. Four axes were needed by the model to complete the optimization, with an average computing time of 0.39 second, with a corresponding mean misclassification error of 10.64%. The linear option could not go beyond that complexity, and we switched to the quadratic option. For the same set size, the solver managed to give instantly better results, with a total mean misclassification error of 7.38% while using one axis only. Then, as we increased the size of the training set, two axes were needed by the solver to complete the optimization process for the equally-distributed set sizes of 200, 500 and 1,000. Impressive results were delivered, as the mean misclassification error kept on significantly decreasing with the growing set size; from 7.31% for a size of 200, to 4.2% and 2.67% for sizes of 500 and 1,000 respectively. However, the solver needed significantly more time to reach its objective as the initial set grew. Specifically, the set size of 1,000 scenarios required an average time of around 15 minutes, with a standard deviation of 14 minutes, whereas the smaller training sets required mere seconds. This contrast in time consumption shows the growing complexity of

this methodology when slightly increasing the set size of scenarios the model is based upon. Furthermore, when we tried this model on bigger initial sets, no successful results could be obtained in a reasonable amount of time, at least not meeting the industrial specifications.

As a conclusion, the Global MILP approach for a border model was demonstrated to be effective on small training sets, especially when tested with a commercial solver which is more robust in dealing with the complexity of the approach. Nonetheless, the industrial requirements lead us to try to adapt the mathematical model in order to reduce its intrinsic complexity, with the aim of delivering a border model that would not be dependent on a commercial solver while nevertheless meeting the industrial specifications. It should also be considered that this new border model should be able to support much larger set sizes, in order to be able to tackle other real use cases of higher dimension. This is why we propose the following approach that focuses on detecting one axis at a time, as detailed in next Section.

6.3.3 Iterative MILP Approach

This second approach is made of two building blocks. Similarly to the Global approach presented above, the **One Axis** method, first component of the Iterative approach applies to cases where the NG area is made of a single connected component, but discovers the axes in an iterative manner, one after the other. The second component uses **clustering** to handle the general case of multiple connected components, using the One Axis method to handle each cluster in turn. When the One Axis fails, a new step of clustering is recursively called.

The **One Axis step** focuses on decomposing the problem of finding all border separations simultaneously into smaller problems that detect each axis one at time via successive iterations. Technically, the axes are detected by minimizing the error produced by the wrong classification of G scenarios. After the first axis has been identified, some scenarios are correctly classified as G by this axis, and other scenarios are classified as NG. The latter scenarios can be a mixture of G/NG scenarios awaiting to be separated as illustrated once more on Figure 6.7. Before moving on to detect the next axis, the idea is to eliminate the G scenarios that have been correctly classified by the first axis. In fact, as explained previously, these scenarios are the ones that have the main influence on the axis equation, and since they have been correctly classified now, they can be eliminated, thus reducing the number of constraints in the model, i.e. the model complexity. Throughout the iterations, the model continues to add consecutive border separations until no more G scenario is left in the input space. This One Axis step has been tested with both solvers, **lp_solve** and **Gurobi**.

The **Clustering step** of the Iterative Model approach focuses on the cases where there are multiple distinct NG zones in the input space. So far, we have only considered a single NG area, that can be represented by a fixed model tree and the corresponding rigid structure of equations. This structure needs to be set before launching the optimization, which gives no flexibility to the approach. The idea now is to perform clustering on the NG zones when the One Axis method is unable to add another axis that correctly separates some G scenarios. After clustering, the One Axis solver is launched on each cluster separately.

We will begin by detailing the One Axis approach, presenting results on test cases with a single NG zone before moving to the clustering step.

Iterative One Axis Algorithm Because one axis is sought at a given iteration, and with similar notation than for Equations (6.3) and (6.4) (though N and N_G now relate to the current training set, that will be different at each iteration of the One Axis algorithm), the constraints now are:

$$\forall i \in [1, N_G], \quad \sum_j (\alpha_j^+ - \alpha_j^-).X_{ij} + \epsilon_i \geq \delta \tag{6.7}$$

$$\forall i \in [N_G + 1, N], \quad \sum_j (\alpha_j^+ - \alpha_j^-).X_{ij} \leq -\delta \tag{6.8}$$

where ϵ_i represent the error for each G scenario, which is required to be strictly positive (and lead to reverse the signs of both equations for simplicity of notations): While the NG constraints are still required to be exactly met (with tolerance δ), errors are allowed for the G scenarios.

These equations (6.7) and (6.8) are much simpler than Equations (6.3) and (6.4) related to the Global MILP model. Indeed, we are now looking for one single axis, hence the variable k has disappeared and there is a single constraint per scenario. Also, there is no need any more for the Boolean variables to indicate which axis separated which scenarios. On the other hand, the small δ constant is still relevant to avoid getting trivial solutions that do not respond to the problem objective, and has been also added to the G constraints after thorough testing.

Finally, errors on the G scenarios are admissible, but should nevertheless remain as small as possible: they also have been added to the objective function, to be minimized:

$$\min \quad \sum_j (\alpha_j^+ + \alpha_j^-) + \lambda \sum_i \epsilon_i \tag{6.9}$$

Because in fact we now have two objectives, a new hyper-parameter had to be added, the positive constant λ, that will need to be set by the user and will tune the balance between the classification error on the G scenarios and the coefficients of the axis under scrutiny. However, its value is problem-dependent, and maybe even iteration-dependent. For instance, if we apply the One Axis algorithm to the test case #2 of the first model with the linear option, we obtain the results shown in Figure 6.14 with a weight value of 10.

We can clearly observe here the way the iterative One Axis algorithm proceeds. After the first axis is found, the G scenarios which are on the upper side of the axis, i.e., which are correctly classified, are removed before continuing with the next iteration. Then, the solver will be asked to find the next axis to separate the remaining G scenarios, and will continue doing so until there is no G scenario left. We can also see that the problem was solved here

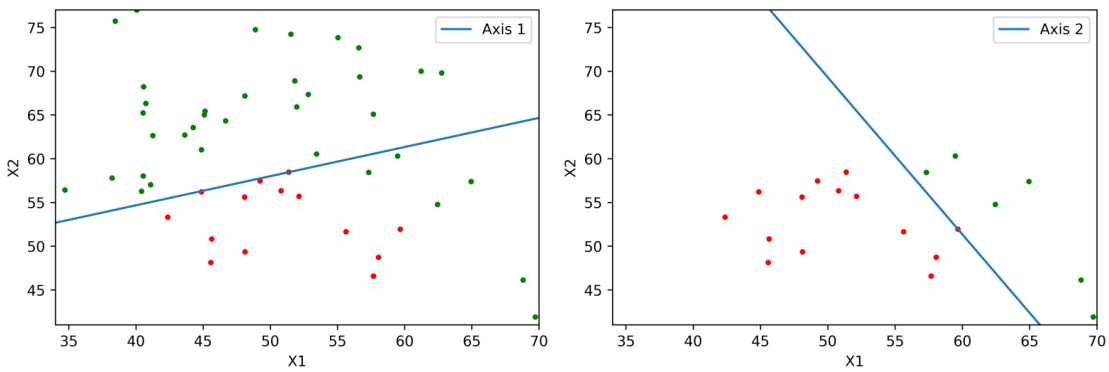

Figure 6.14: One axis edition of MILP realized on test case #2 consisting of 50 scenarios including G scenarios (green) and a single NG disk zone (red). First axis (left) and second axis (right) are found consecutively by the solver with $\delta = 0.05$ and $\lambda = 10$.

with two linear axes instead of three as done previously with the Global model (Figure 6.10). This is explained by the fact that we have written the constraints equations in a way that prohibits projection errors for the NG scenarios (by adding the classification errors ϵ_i to the G constraints equations). Thus, the axes will pass by the NG scenarios, and this is why the solver could solve the problem using two axes only. The other test cases where one axis is enough to complete the separation process were also successfully solved, and the results are similar to the ones obtained with the Global approach. However, bigger values of the constant λ could be needed for obtaining good results. For instance, for the test case number #3, we tried several values of λ. Figure 6.15 shows the results obtained for $\lambda = 0.1, 1, 10, 100$.

For the values of λ equal to 0.1, 1, and 10, the solver is unable to detect the second axis. The weight is not high enough to effectively impact the objective function. As for the value of 100, the solver is again able to solve the problem correctly, especially that is was able to detect the first circular axis (in contrast with the previous values of λ tested). However, it still outputs a very elongated ellipse as the second axis. We decided to settle for this problem on a value set to 1000 for λ, whose results are illustrated in Figure 6.16. Here, the second axis is less elongated than the case with $\lambda = 100$, while the first circular axis remains correctly identified. Therefore, the λ parameter becomes a new hyper-parameter to effectively tune for this approach to deliver successful results.

The solver managed to detect the two axes that delimit the NG scenarios apart from the G scenarios, although it was unable to find the inner circle due to lack of points, as discussed in Section 6.3.2. Nevertheless, it is clear that the iterative One Axis approach is much more flexible than the Global approach because there is no need any more to set the number of axes a priori, at the beginning of the optimization process, and adjust it by trial and error. The One Axis approach iteratively identifies the need for an additional axis, until a complete solution is delivered. The price for that, however, is the additional hyperparameter λ added

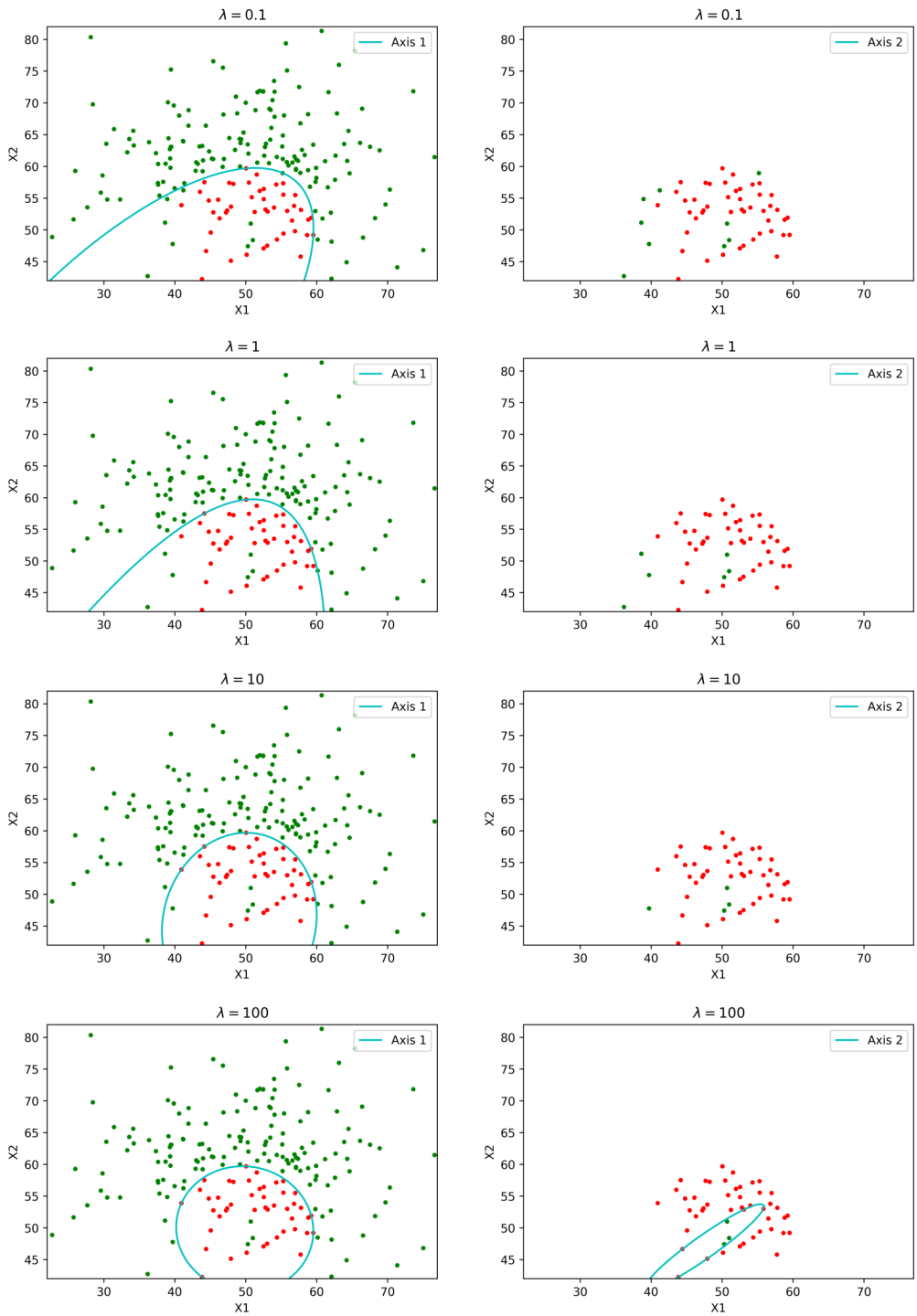

Figure 6.15: One axis edition of MILP realized on test case #3 consisting of 200 scenarios including a single NG zone (red) located between two concentric circles separating them apart from the remaining G (green), while changing the value of the λ hyperparameter ($\delta = 0.05$).

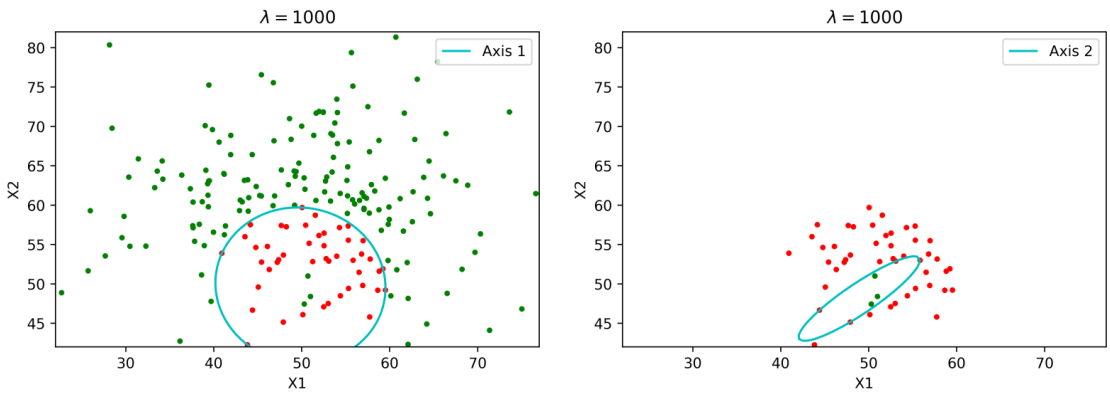

Figure 6.16: One axis edition of MILP realized on test case #3 consisting of 200 scenarios including a single NG zone (red) located between two concentric circles separating them apart from the remaining G (green). First axis (left) and second axis (right) are found consecutively by the solver with $\delta = 0.05$ and $\lambda = 1000$.

to the model.

Iterative Clustering Now that we have shown that the methodology is effective for a single connected NG area, let us move to the second step of this approach, the clustering, in the case where the NG area has more than one connected component, that cannot be solved directly with the approaches proposed so far. As said, the idea is to use a clustering technique on the remaining scenarios as soon as the solver is unable to find another axis that correctly separates at least one G scenario from the other ones. The goal of using clustering is to distribute the NG zones into different subproblems and to tackle them independently: The One Axis approach is applied to each cluster, and the process is repeated until all G scenarios are correctly classified. Hence, not only the tree representation is built step by step, without specifying it in advance, but also much more complex trees can be built (see e.g., Figure 6.19) thanks to the intertwining of clustering and solving.

The Process In order to implement this hybrid approach, several classes have been created in Python, as shown in Figure 6.17.

A parent class named **Node** is first created. It contains two attributes, left leaf and right leaf, and a method called *predict()* to evaluate whether the scenario is G or NG. Then, two other classes called **Axis** and **Cluster** are created as inherited classes from the **Node** class. On one hand, the **Axis** class defines its right leaf as a new **Node** object instance and its left leaf as 0. This means that, when an axis has been used, the node splits to only G scenarios correctly classified to the left, and the rest of the optimization continues to the right. On the other hand, the **Cluster** class defines its right and left leaves as new **Node**

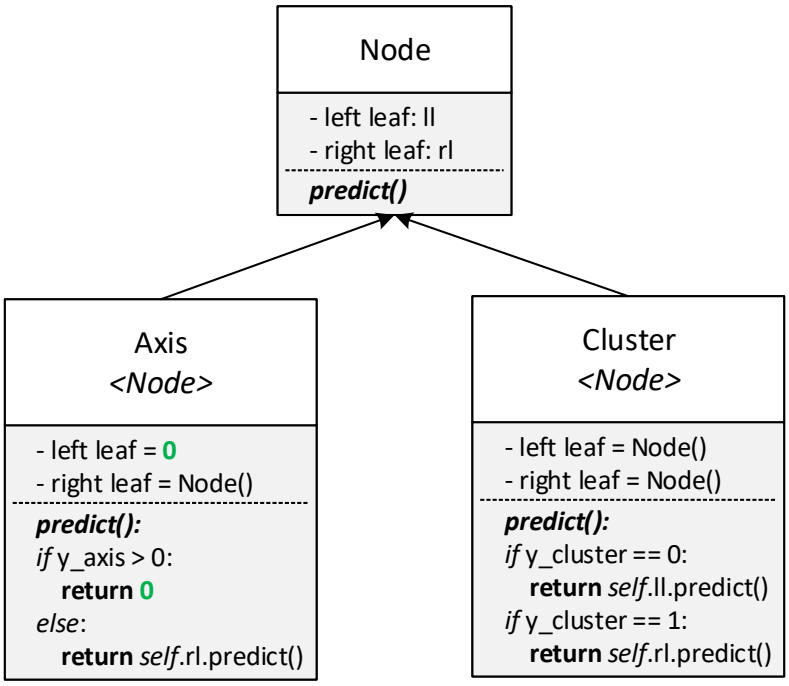

Figure 6.17: Classes created for the tree representation of the Iterative approach of MILP: the Node class is parent to the Axis and Cluster classes.

object instances since no G/NG separation has occurred. The optimization then resumes in both leaves separately.

As to the *predict()* function, it should ultimately output a value of 0 if the scenario is classified as G and 1 if it is NG. Each derived class applies it to the scenario to be predicted. For the **Axis** class, the following value is first calculated: $y_{i,axis} = \sum_j \alpha_j.X_{ij}$. If it is strictly positive, then the outcome is set to 0 to indicate a correctly classified G scenario. Otherwise, the outcome is redirected into applying the prediction on the right leaf of the axis in a recursive fashion. As for the **Cluster** class, we first determine to which subgroup, left or right, the scenario belongs to after applying the cluster on it. Then, depending on the outcome, we redirect the recursive *predict()* function to the left or to the right leaf. In that way, we are building the tree representation step by step to ultimately output the correct prediction when the method has terminated and the tree has been completed.

As for the clustering technique, we have chosen the **k-means** clustering which partitions the observations into clusters depending on the nearest means, also called cluster centers or centroids. Plus, we fit the clustering only on the NG scenarios to ensure that we partition the corresponding areas into different clusters. In this work, we set $k = 2$ to test this approach with 2 clusters created each time. The main function of this whole methodology **milp_one**

is briefly summarized in Algorithm 6.

Algorithm 6: MILP One Axis Main Function: milp_one(training set)

```
if all scenarios in training set are NG then
    return 1
else
    • Build MILP model based on Equations (6.7, 6.8, 6.9) and launch solver
    • Retrieve errors ε_i of Equation (6.7)
    if there is at least one G scenario correctly classified with ε_i = 0 then
        • Create new Axis instance: New_axis
        • Create subset by eliminating the scenarios that have been correctly
          classified is G
        • Launch recursively the function on the right leaf of the Axis instance:
          New_axis.rl = milp_one(subset)
        • return New_axis
    else
        • Fit k-means clustering on the NG scenarios
        • Create new Cluster object instance: New_cluster
        • Create two subsets corresponding to each cluster obtained: left_cluster and
          right_cluster
        • Launch recursively the function on the left and right leaves of the Cluster
          instance: New_cluster.ll = milp_one(left_cluster) and New_cluster.rl =
          milp_one(right_cluster)
        • return New_cluster
    end
end
```

This function is developed to be used recursively in order to build the tree representation as the optimization proceeds. When it is called on a set of scenarios, it checks first if all the scenarios are *NG*. This means that the problem has been resolved completely, and the function returns the value 1. If this is not the case, the MILP model is built based on the constraints and objective functions of Equations (6.7, 6.8, 6.9). We then examine the values of the errors found in the constraint equations of the G scenarios. If at least one of these values is equal to 0, this means that the model was able to find an axis that correctly classifies at least one scenario as G by separating it from the rest of the scenarios. A new **Axis** instance is thus created, and the scenarios with zero errors are eliminated to create a new subset. Since the left leaf of the **Axis** object represents these G scenarios and has a prediction value

of 0, the main function **milp_one** is then called on the new subset through the right leaf of the object to pursue the border model process. The function returns the **Axis** object in an effort to build the tree representation step by step.

However, if none of the error values is equal to 0, this indicates that no axis has been found by the solver that could correctly classify at least one scenario as G, meaning that multiple NG distinct areas are present in the input space. We then perform k-means clustering by fitting it into only the NG scenarios to capture the NG areas centroids more effectively. A new **Cluster** object is hence created, and the two clusters that emerge help in defining the two subsets derived from this clustering. Finally, the main function **milp_one** is called on both clusters through the left and right leaves of the **Cluster** object, and the function returns the **Cluster** object to pursue the tree construction.

To better visualize this methodology, we tested it on two new test cases #4 and #5 containing 500 scenarios with 2 and 3 distinct disk NG areas respectively. Let us begin with test case #4, with 2 NG zones. The final tree representation can be seen in Figure 6.18, whereas Figure 6.19 shows the visual step-by-step unfolding of the algorithm.

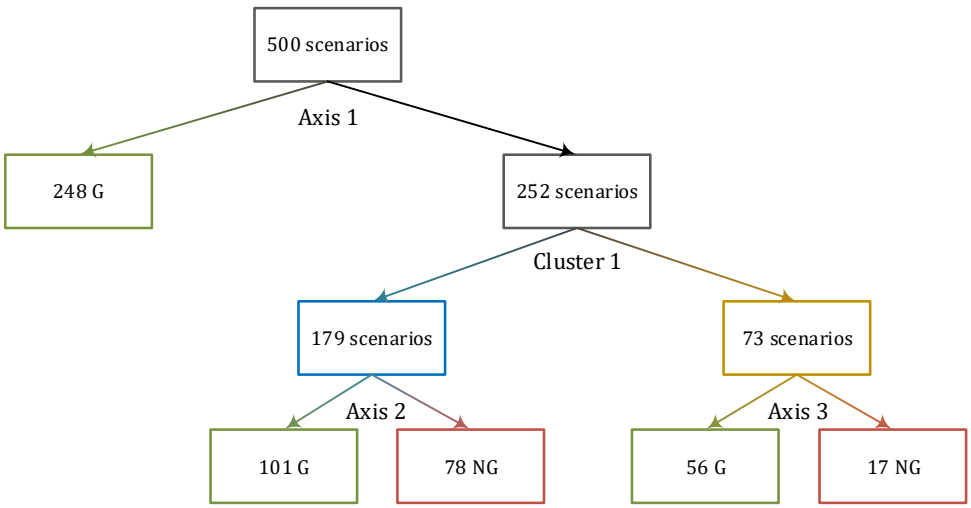

Figure 6.18: Final tree representation of the results of the Iterative MILP approach applied on test case #4.

Test case #4 consists of two NG disk areas as seen in graph 1 of Figure 6.19 for a total of 500 scenarios. The solver first finds a quadratic axis that classifies correctly 248 G scenarios out of the 500 (graph 2). The remaining 252 scenarios are then evaluated by the model for a possible axis (graph 3). However, the solver was unable to find such an axis, which leads the algorithm to do one k-means clustering step. The scenarios are divided into 2 clusters of 179 and 73 scenarios respectively (graph 4). Notice that the resulting centroids are indeed the centers of the NG areas. The algorithm begins by examining the first cluster (graph 5).

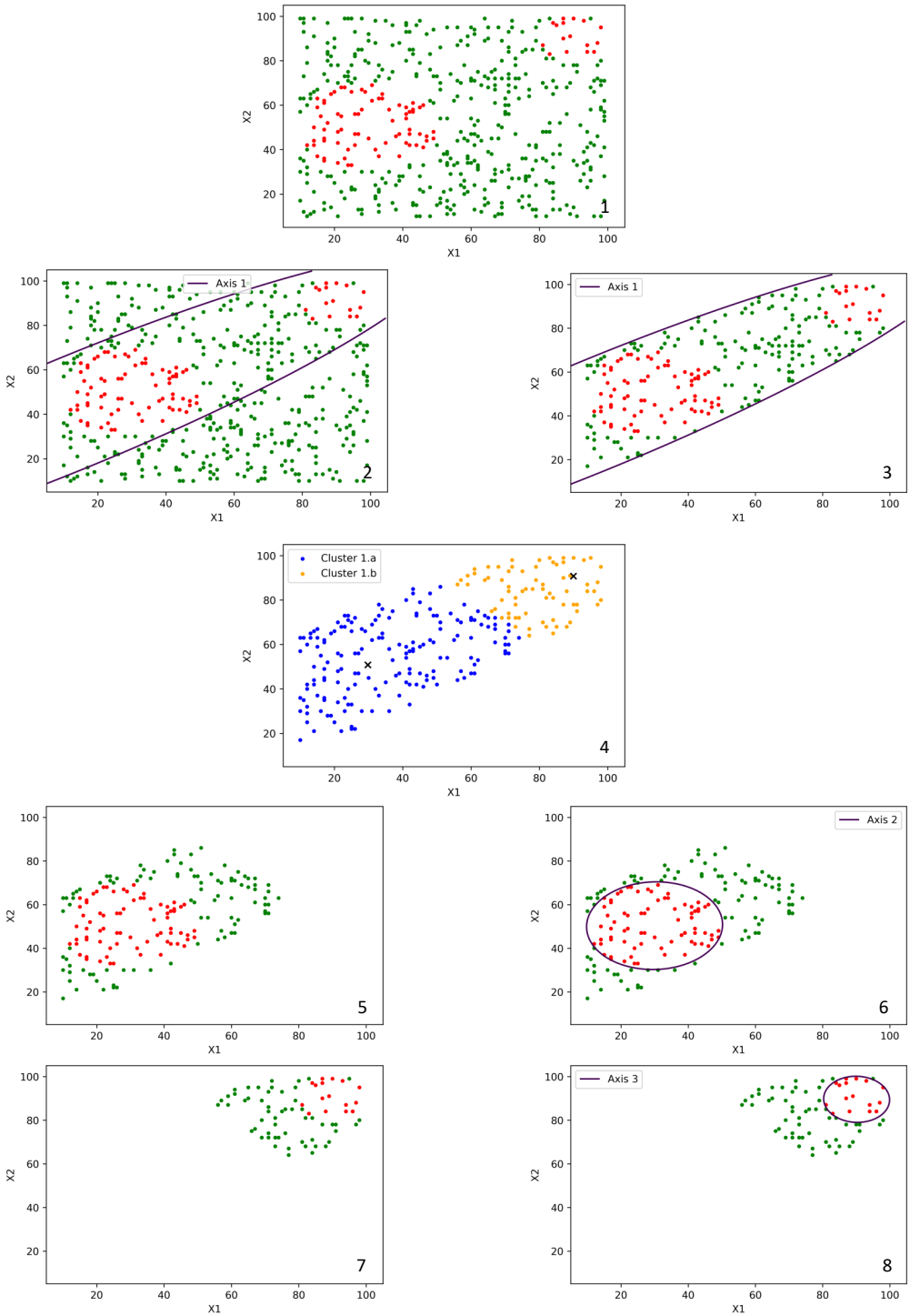

Figure 6.19: Step-by-step results of the Iterative MILP approach applied on test case #4 containing two distinct *NG* areas.

The solver succeeds in separating all the 101 remaining G scenarios by a single axis from the 78 NG scenarios (graph 6). Finally, the algorithm handles the second cluster of 73 scenarios (graph 7), where the solver succeeds in correctly classifying all 56 G scenarios apart from the 17 NG scenarios (graph 8). Problem solved.

We now turn to the more complex problem of test case #5 where three NG failed disk areas are present in the search space. Figure 6.20 illustrates the final tree representation, as the border model succeeds in resolving the problem while showing the different axes and clusters added on its way in Figure 6.21.

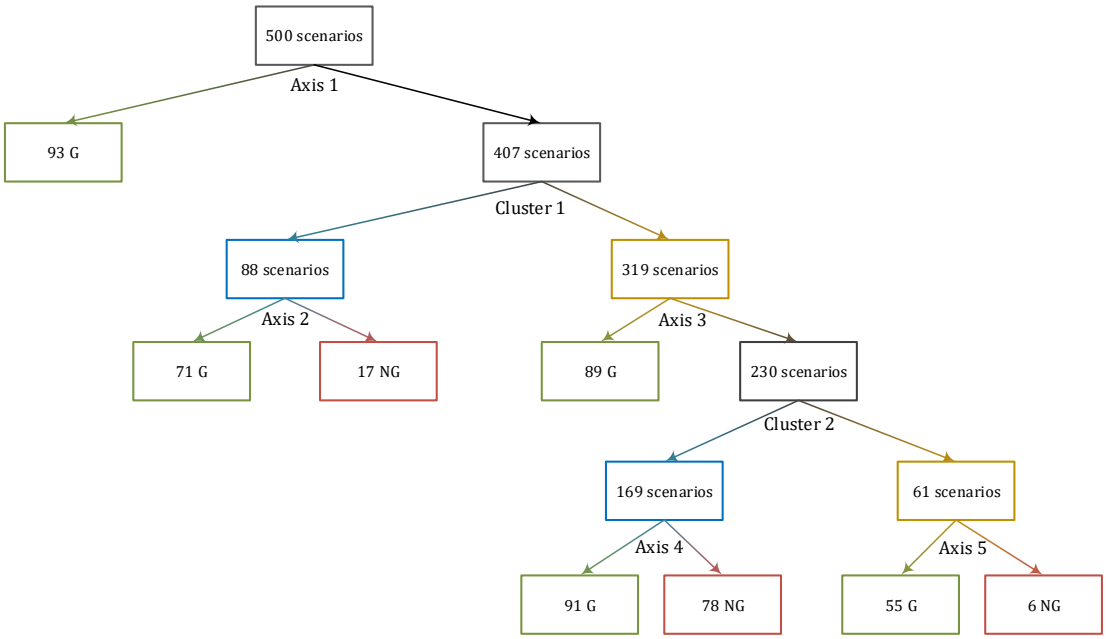

Figure 6.20: Final tree representation of the results of the Iterative MILP approach applied on test case #5.

Graph 1 of Figure 6.21 shows the three distinct NG areas in the search space, and the 500 scenarios of the initial training set. The solver succeeds in identifying a first axis that separates 93 G scenarios from the remaining 407 scenarios (graph 2). Because it was unable to continue due to the presence of the three NG areas, a first clustering is realized (graph 3) that divides the 407 scenarios into two clusters of 88 and 319 scenarios. In the first cluster, there is only one NG area of 17 scenarios which is successfully encircled by a second axis identified by the solver (graph 4). As for the second cluster, a third axis is identified, that delimits 89 correctly classified G scenarios from the remaining 230 scenarios (see graph 5). Then, another clustering needs to be launched to create two clusters of 169 and 61 scenarios, each containing one NG area (graph 6). Ultimately, the solver completes the problem by adding the fourth and fifth axes which encircle the 78 and 6 NG scenarios of each cluster

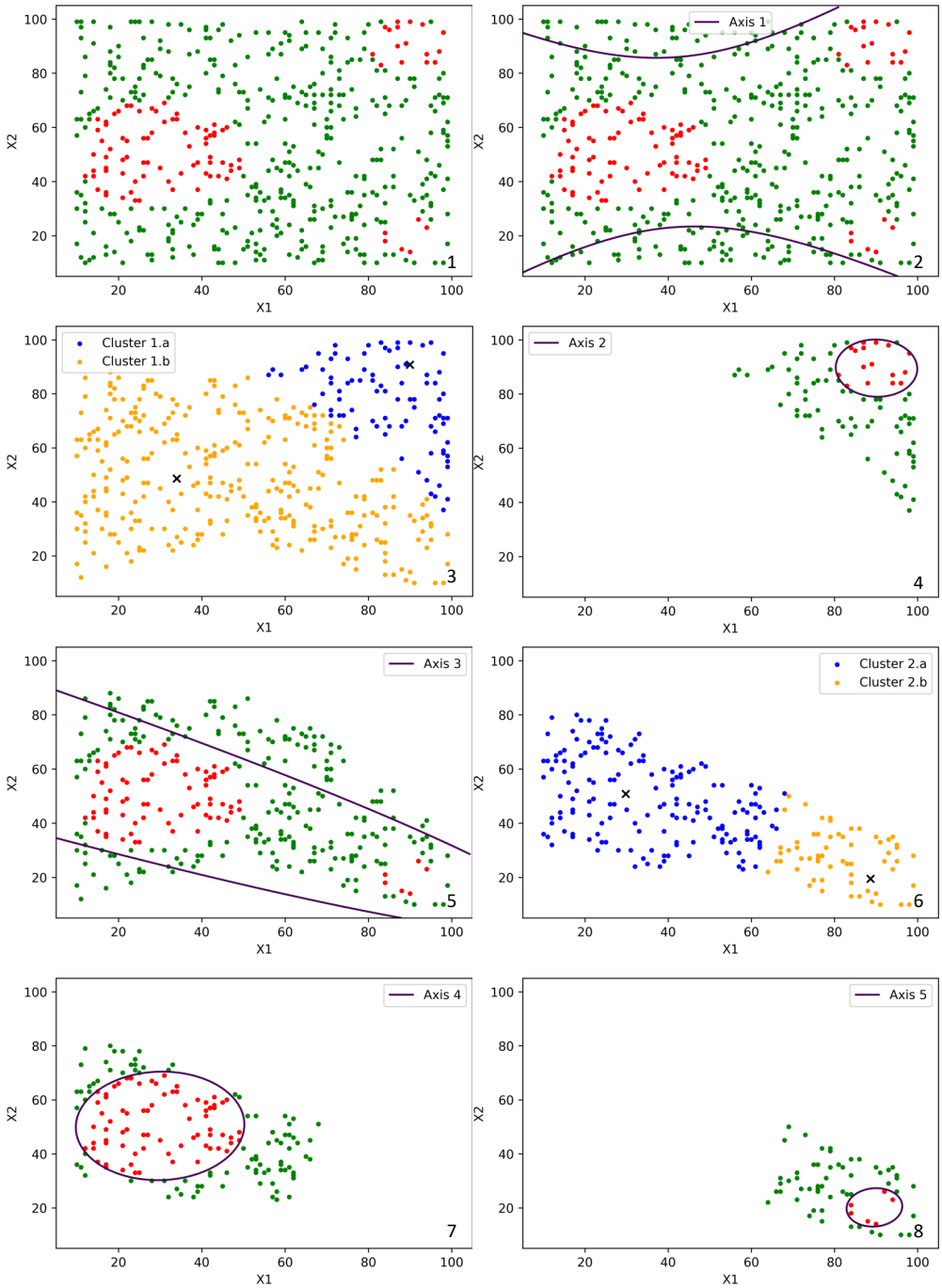

Figure 6.21: Step-by-step results of the Iterative MILP approach applied on test case #5 containing three distinct NG areas.

(graphs 7 and 8).

The proposed methodology demonstrated its effectiveness on these two test cases. It is worth noting that the algorithm had to create two clusters for the test case number 5 with three distinct NG areas in order to divide them and solve the problem. A possible enhancement would be to increase the number of clusters, though this raises the issue of how to determine which number of clusters is optimal, which would require the remodeling of the **Cluster** class.

Finally, the method is applied on the real use case NHTSA 13 to evaluate its performance, after all inputs have been normalized in order to improve the numerical stability. It is applied to different training set sizes, while showing the number of axes used, the number of clustering steps used, the overall time consumed by the solver, and the corresponding misclassification errors computed on the full grid of 470,587 scenarios. Similar to the Global MILP approach, 11 independent runs were performed with 11 different training sets for each training set size considered, and Table 6.3 reports the mean and standard deviations of the results.

Table 6.3: Results of the Iterative MILP algorithm on the NHTSA 13 use case.

Set size	**Linear / Quadratic**	**Number of axes**		**Number of clustering steps**		**Time (sec)**		**Misclassification error (%)**	
		Mean	SD	Mean	SD	Mean	SD	Mean	SD
100	Linear	5.0	1.48	2.0	1.13	0.35	0.11	10.61	2.43
200	Linear	8.91	2.84	3.36	1.67	0.68	0.17	7.69	1.76
500	Linear	17.36	2.9	8.55	2.23	1.73	0.29	6.19	1.03
1000	Linear	33.18	3.71	17.36	3.05	3.51	0.31	5.15	0.48
5000	Linear	114.45	9.22	76.55	5.02	17.88	0.67	2.82	0.14
10000	Linear	193.45	11.16	137.27	7.39	34.94	0.56	2.19	0.08
100	Quadratic	1.0	0.0	0.0	0.0	0.13	0.01	8.96	1.59
200	Quadratic	1.18	0.57	0.0	0.0	0.24	0.1	7.08	1.94
500	Quadratic	3.09	1.0	0.27	0.45	1.23	0.37	5.56	0.85
1000	Quadratic	5.09	1.5	1.45	0.66	3.38	0.59	4.08	0.65
5000	Quadratic	19.09	1.88	6.18	1.11	29.42	1.71	2.2	0.25
10000	Quadratic	37.82	3.04	13.36	1.23	68.92	2.43	1.78	0.09

The main advantage of this approach, compared to the Global MILP modeling, is that is was able to successfully complete all experiments in a reasonable amount of time (under the one minute time frame). The experiments were made for training sets of 100, 200, 500, 1,000, 5,000 and 10,000 scenarios, each containing an equal number of on-border and off-border scenarios. Plus, they were tested with the linear and quadratic options to evaluate the difference in numbers of axes and 2-means clusterings used along the optimization process. Lastly, the misclassification error is calculated for all 470,587 scenarios of the use case full grid. For each scenario, we evaluate where it fits in the tree representation, and compare the predicted output to its real label.

As expected, the numbers of axes and 2-means clustering steps are much larger for the

linear option than for the quadratic option, and, for both options, these numbers increase with the size of the training set:. For instance, the set sizes of 100, 1,000 and 10,000 need in average 5, 33 and 193 linear axes with 2, 17 and 137 clustering steps respectively, whereas they only require in average 1, 5 and 38 quadratic axes along with 0, 1 and 13 clustering steps to reach the same objective. At the same time, for both options, the total mean misclassification error tends to decrease when increasing the set size. Nonetheless, the quadratic option achieves a better accuracy than the linear option for all experiments. For example, the solver reached linearly 10.61%, 5.15% and 2.19% of total mean misclassification error for the sets of size 100, 1,000 and 10,000 respectively, whereas it computed 8.96%, 4.08% and 1.78% of total error for these same sets using the quadratic option.

Comparing now Tables 6.3 and 6.2, the Iterative approach seems to be clearly more effective that the Global approach, mainly because it was able to reduce its intrinsic complexity by offering a solution that breaks down the problem into step-by-step tree building. Plus, it can handle the presence of multiple NG distinct areas thanks to the clustering steps without the need of manually adjusting the tree or choosing the number of axes in advance. This adaptive construction of the tree representation gives more flexibility and results in a broader application range thanks to the reduction of the mathematical model complexity. However, the performance of the Global approach is better than that of the Iterative approach in terms of accuracy for the case of 1,000 scenarios in the training set (2.67 ± 0.42 vs 4.08 ± 0.65) - though it should be kept in mind that the Global approach needed between 10 and 30 times more computing time (with a very large standard deviation). However, only 2 axes were needed without clustering, against 5 for the Iterative solution, making the Global approach much more explainable than the Iterative one.

This observation could be a hint that clustering steps should be delayed after more useful axes have been found in a given One axis step, based on some well-defined criterion. Another hint is given by taking a closer look at graphs 2 and 5 of Figure 6.21: while the axes built by the algorithm correctly classify a large number of G scenarios, there are other G scenarios that could still have been also classified before performing the next clustering step. This is due to the fact that the algorithm finds it more beneficial to perform clustering than to generate more axes and separate more G scenarios from the NG ones. This also goes against the industrial specifications for explainability: The separation of scenarios into clusters is less explainable, and hence clustering should be used only if there are no G scenarios left to be classified. Therefore, in the example shown, a handful of axes could still have been detected, leading to a better close-up of the NG areas, before performing clustering. A possible solution could be to add more variables and Booleans to force the model to go in this direction, but it would increase the model complexity. Instead, we decided to develop a further model: the Greedy model is presented in next Section.

6.3.4 Greedy MILP Algorithm

The Greedy Model approach draws heavy inspiration from the Iterative Model. Its main objective is to force the algorithm to find all possible axes that delimit the NG area and only switch to clustering when there is absolutely no G scenario that could be further correctly classified. The only other difference with the Iterative Model is the method used to find a certain axis, but the general idea of detecting axes step-by-step with clustering remains the same. The general picture of the Greedy approach is described in Algorithm 7, and will be detailed in the following paragraphs.

Algorithm 7: MILP Greedy approach for finding a single axis

Init: randomly pick p scenarios s.t. an axis that passes through these points exists
- Compute the $\vec{\alpha}$ coefficients of an axis passing though the current p scenarios
- Compute the projections of the current p scenarios on this axis
- Compute the value of the objective function used to evaluate the axis (Equation (6.10))
- Change the p current scenarios and **loop**

Return the axis that minimizes the objective function while keeping all NG scenarios on the same side of that axis

First, we begin by picking p NG scenarios (p being typically equal to n, the dimension of the use case), in such a way that an axis that passes through them exists. For the two-dimensional test cases, we chose to draw the first scenario randomly, whereas the second one is picked following a probability distribution in which the probability to choose one point is proportional to its distance with the first scenario, in an effort to chase farther points.

The next step is to make sure that the axis exists and compute its coefficients $\vec{\alpha}$, which amounts to solving a homogeneous linear system $A\vec{\alpha} = 0$, where A is the matrix containing the coordinates of the chosen scenarios of dimension $p \times (n+1)$ as the coordinates are augmented with $X_0 = 1$ to account for the constant coefficient of the axis equation; And $\vec{\alpha}$ is the vector of the unknown axis coefficients. If $p = n$, the system is underdetermined. Furthermore, we need to avoid the trivial solution $x = 0$. Hence we solve this system by Singular Value Decomposition (SVD) on the matrix A.

Next, the projection of all the scenarios on the axis: $\sum_j \alpha_j X_{ij}$ are calculated to lay the ground for the computation of the objective function to be minimized for the choice of the best axis, which is defined as follows:

$$\sum_{i \in NG} N_G^2 \left| \sum_j \alpha_j . X_{ij} \right|^+ - \sum_{i \in G} \left| \sum_j \alpha_j . X_{ij} \right|^+ \tag{6.10}$$

The first part of this objective function corresponds to the sum of the projections of the NG scenarios that had a positive projection on the axis, meaning that they were incorrectly

classified. The second term is the sum of the positive projections of the G scenarios that were correctly classified. The axis with the minimum criterion is chosen, which means that we are looking for an axis that can simultaneously minimize the NG scenarios misclassification and maximize the G scenarios classification. A weight, equal to the square number of G scenarios, has been added to the first term of this equation to give more importance to the NG misclassification when scoring the axes.

After calculating the criterion for the initial p scenarios, we are set to explore the input space to detect the best axis candidate. In the case $p = n = 2$, the idea is to keep one scenario of the current pair and to do an exhaustive search among all other possible NG scenarios which were wrongfully classified by the axis formed by the initial p points. For each combination of scenarios, the corresponding axis coefficients are computed, as well as its objective function. The axis that minimizes the objective function is retained, and is returned if it classifies all NG scenarios on the same side. Otherwise, it repeats the whole procedure while considering the p scenarios of the best axis obtained as initial scenarios. Note that in particular cases where the algorithm is unable to find an axis with better criterion than its initial axis, other initial scenarios are drawn and the procedure repeats.

Finally, after finding the best axis that meets all requirements set, the G scenarios correctly classified by this axis are discarded, and the algorithm proceeds to the next axis detection, in a iterative way similar to the One axis MILP algorithm (Section 6.3.3). Nonetheless, if the chosen axis was unable to classify a single G scenario, then, and only then, the model performs a 2-means clustering step on the NG scenarios.

Furthermore, a few minor algorithmic improvements were added to ensure that the algorithm always manages to detect a meaningful axis: i) axes that already exist in the tree representation become taboo, and cannot be used again; ii) to avoid that the axis search is only performed in a single direction starting from the initial scenarios, the coefficients of any chosen axis are multiplied by -1 and the search is repeated for the same axis in the other direction.

In order to illustrate the proposed Greedy approach at work, it was applied to test case #5, containing the three NG distinct zones, using the linear option. The main steps are presented in several Figures below.

Figure 6.22 shows the steps taken by the Greedy MILP approach until it reaches its first clustering step.

In this figure, we can see the main difference between the Greedy approach and the previous Iterative approach. For instance, the graph 2 in Figure 6.21 shows an axis found by the algorithm that was able to classify some G scenarios around the NG areas. However, the model performed a clustering step immediately after that in graph 3. In contrast, for the Greedy approach, graphs 2 to 7 in Figure 6.22 illustrate the different steps to clean all G scenarios around the NG areas, even if the axis is introduced to classify a single G scenario as seen in graph 5. When all these scenarios are eliminated and no possible axis could classify any more G scenarios while simultaneously having all NG scenarios on the same side of the

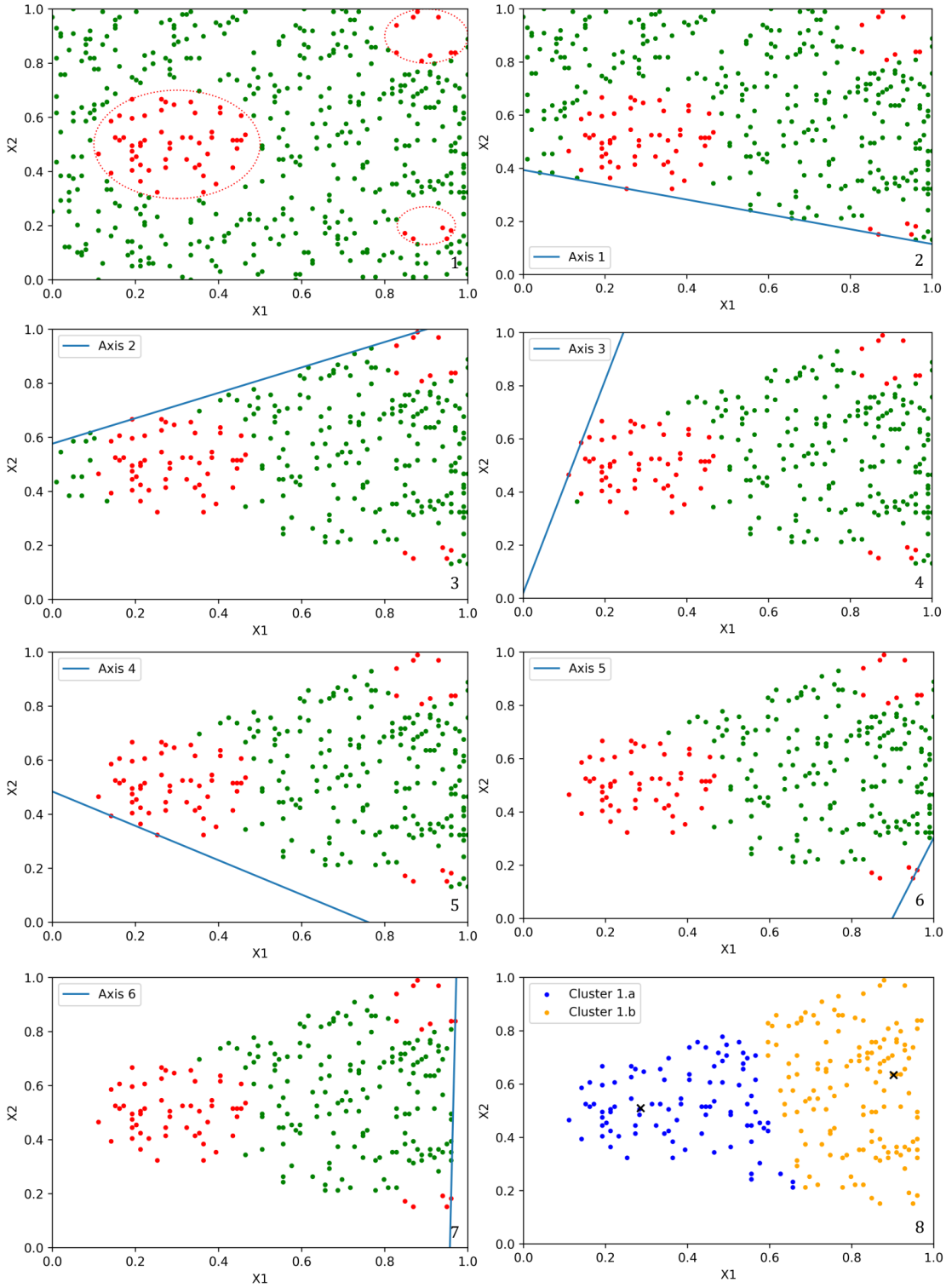

Figure 6.22: Step-by-step results (Part 1) of the Greedy edition of MILP applied on test case #5 containing three distinct NG areas until reaching its first clustering used.

axis, the algorithm turns to clustering (graph 8). This behavior seems to meet the industrial requirements for explainability, implying that clustering should be used as little as possible, in contrast with the Iterative approach.

Nonetheless, as the algorithm has to deal with less and less scenarios, it will ultimately trigger a clustering step even though an axis should be detected instead. This aspect can be seen in the visual continuation of the previous graphs, presented in Figure 6.23.

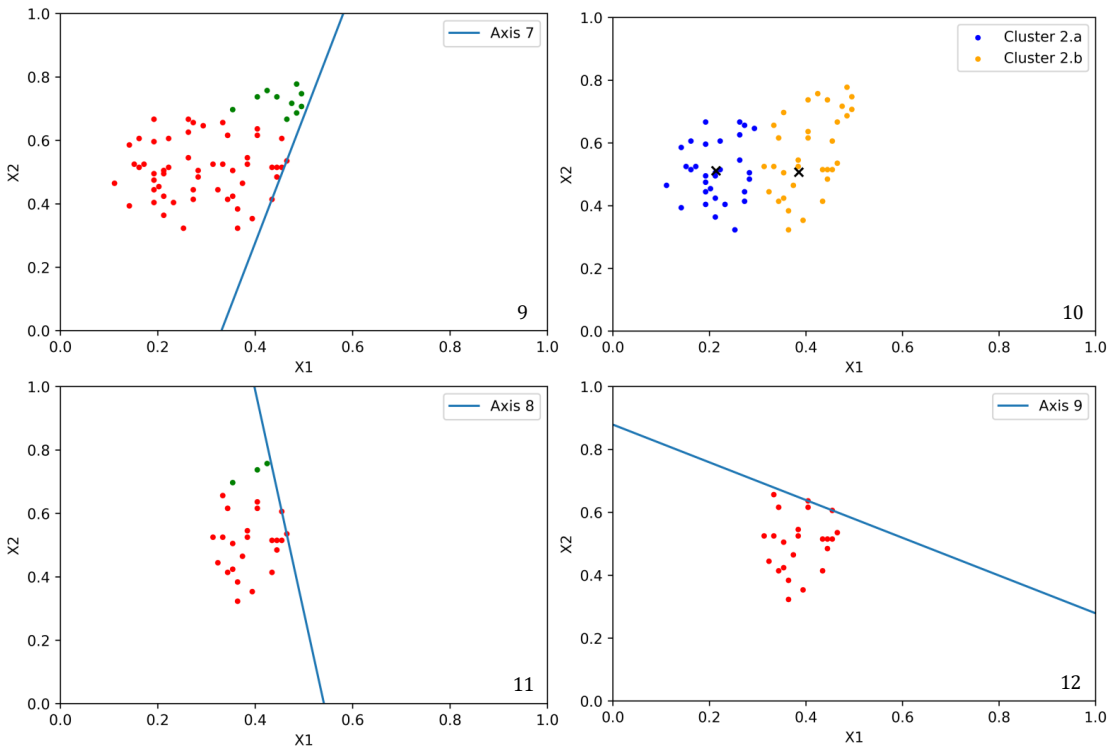

Figure 6.23: Step-by-step results (Part 2) of the Greedy edition of MILP applied on test case #5 containing three distinct *NG* areas after performing its first clustering.

In graph 9, an axis was successfully introduced by the algorithm to the first cluster, and the corresponding *G* scenarios on its right side were discarded. However, in graph 10, the algorithm performed another clustering step before introducing two more axes in graphs 11 and 12 to complete the classification of the *G* scenarios in this cluster. In fact, while the algorithm managed to successfully "encircle" the *NG* scenarios, it used an unnecessary clustering, especially when it could have detected Axis 9 of graph 12 directly after identifying Axis 7 of graph 9, without any additional clustering. This phenomenon happened a second time while completing the problem as seen in Figure 6.24.

When turning to the second cluster, the algorithm found two axes that successfully delimited the two *NG* areas (graphs 13 and 14). Another clustering is unavoidable here to

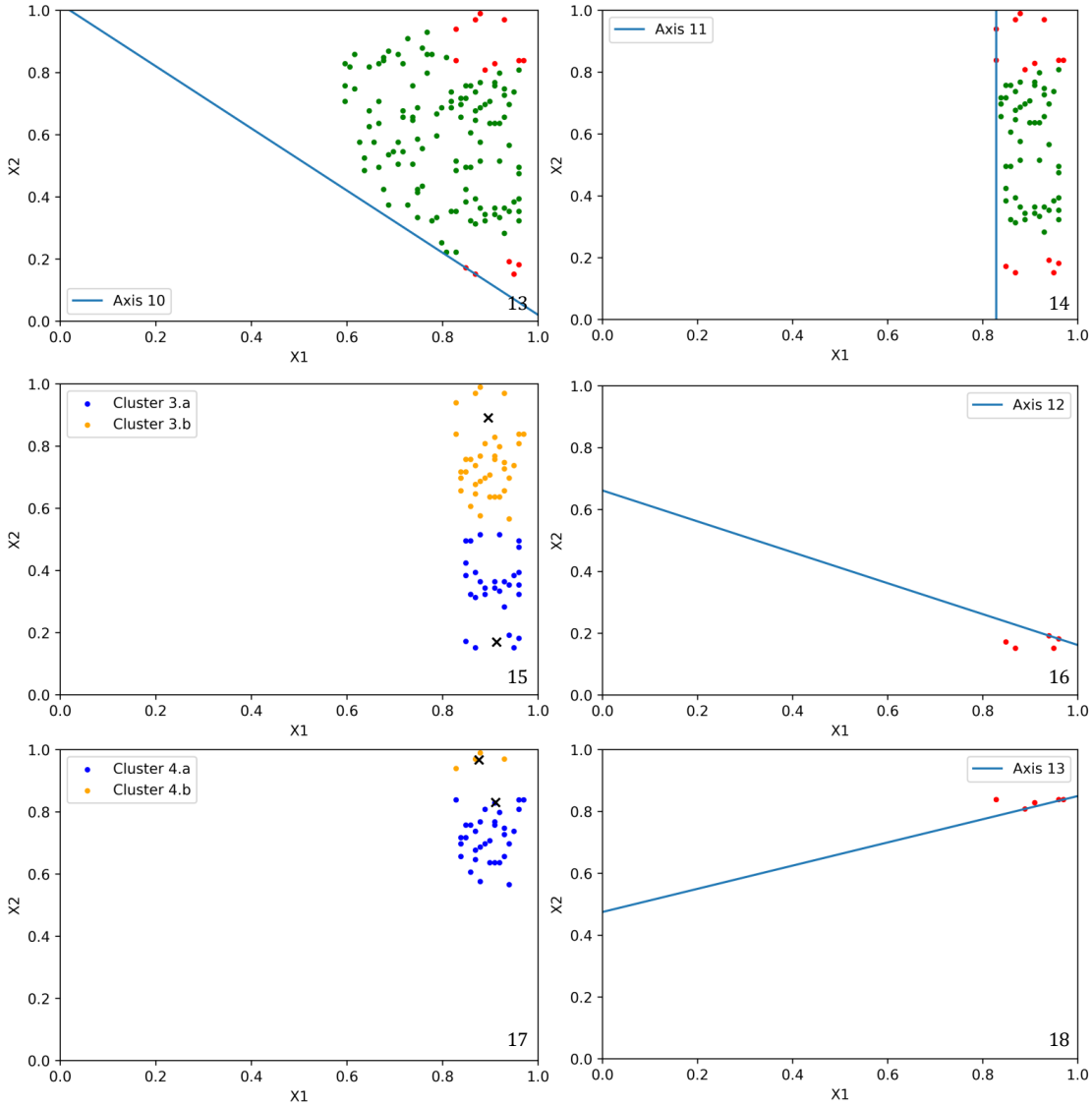

Figure 6.24: Step-by-step results (Part 3) of the Greedy edition of MILP applied on test case #5 containing three distinct *NG* areas until completing the problem.

separate the *NG* areas (graph 15). While the model directly finds an axis to finish the classification in the first cluster, it had to perform yet another clustering in order to find the final axis number 13 that ends the whole process. However, this axis could have been found without the use of this fourth clustering step: Whereas the Greedy border model meets the industrial requirements when sufficient scenarios are available in the input space, it still performs unnecessary clustering steps when less scenarios are present. Nonetheless, since the industrial requirements are met for a big amount of scenarios, which represents most real use

cases, the Greedy MILP approach was chosen at this development stage to be enhanced and industrialized for greater performance and broader application range, namely for use cases of larger dimension, more representative of actual use case, and all further developments of the Greedy approach were outsourced to EURODECISION, one of the top French companies specialized in mathematics and resource optimization.

In addition to enhancing the model, another objective set to EURODECISION is to be able to launch the model symmetrically for the G scenarios. In fact, the model developed so far produces axes that pass between NG scenarios. The idea is then to repeat the procedure for the G scenarios. In that way, we will obtain two tree representations which will create a sort of unknown zone between them. This unknown zone represents the border uncertainty between the G and the NG scenarios. Then, by introducing new points in this uncertain zone, one can hopefully improve the localization of the border, reaching out the long term goal of an active learning procedure.

After some months of development, EURODECISION sent a first industrialized Greedy border algorithm. In order to take into account the two tree representations, they updated the way the model can predict the scenarios G/NG evaluations by adding a third state called *Undefined*. The model now returns two trees: the first one predicts G areas while setting the rest as Undefined, whereas the second predicts NG areas while leaving the rest as Undefined. Then, to evaluate the final prediction of the scenarios, the model applies the following Table 6.4.

Table 6.4: Final prediction of Greedy model using the G and NG trees.

Prediction by NG tree	Prediction by G tree	
	Good	**Undefined**
Undefined	Good	Undefined
No Good	Undefined	No Good

If the G tree returns Good and the NG tree returns Undefined, then the final output is Good. Similarly, if the NG tree return No Good and the G tree return Undefined, then the final evaluation is No Good. Finally, if both trees return Undefined, or output contradictory G/NG outputs, then the final prediction is Undefined.

Conclusion on MILP Border Models The delivery of this algorithm by EURODECISION occurred during the final couple of months of this thesis. Upon delivery, EURODECISION presented the application of the border model on some test cases, while noting that the addition of new points between the two trees was not yet realized. Unfortunately, the testing and application of the model were not able to be conducted during the thesis due to the lack of remaining time, and some API issues.

However, the Iterative "One axis at a time" MILP approach introduced and experimented with in Section 6.3.3 remains a very effective approach, that is able to represent the border in a reasonable duration of time and achieves acceptable accuracy on the whole grid of scenarios.

It even scored error values very close to, if not even slightly lower than, the Neural Network approach for the same sizes of training sets, and without using the iterative active learning process described in Section 6.2.1. For instance, the sets of 1,000, 5,000 and 10,000 scored initial misclassification errors of 6.23%, 2.78% and 2.81% by the Neural Network, as compared to 4.08%, 2.2% and 1.78% for the Iterative MILP model respectively.

In the mean time, and even though the MILP approaches fulfill many of the industrial requirements, as argued above, another completely different possible path to explainable and efficient solutions was also tried, in order to explore several possible complementary approaches. It will be introduced and discussed in next Section.

6.4 Genetic Programming Border Models

Genetic Programming (Section 3.2.3) applied to Symbolic Regression evolves analytical models represented by trees. As such, GP models are not black boxes, and can be said to be explainable, provided the bloat phenomenon is controlled, i.e., the size of the tree is not too large.

6.4.1 Methodology

Symbolic Regression starts with a number of fitness cases (examples from the training set, in the ML vocabulary), and aims to minimize the **fitness**, most likely the RMSE, sum of squares of the differences between the fitness case output and the tree output. The user must first define the **operators** (nodes of the trees): for Symbolic Regression, the minimal set is made of the standard binary operators $+, -, *, \backslash$, where $\backslash$ is the so-called protected division, that returns 1 when the denominator is 0 (or very small) to avoid exceptions; However, any other mathematical unary operators can be added, e.g., sine, cosine, exp, ... to the ternary conditional operator (if, condition, branch1, branch2) that returns branch1 if condition is true, branch2 otherwise. The **terminals** (leaves of the trees) are generally made of the variables of the problem and the real-valued constants that are set at initialization time and usually stay fixed ever after.

The user must also define the variation operators: crossover, very likely to be the standard sub-tree exchange (see Section 3.2.3), and mutation, where point mutation (replacement of a subtree with a randomly generated subtree) is the most frequently used mutation; The initialization procedure (the most popular today is the **ramped half-and-half**); And the selection operators (usually this amounts to choose the size of the tournament selection). Last but not least, the population size and the maximal number of generations complete the set of hyper-parameters.

6.4.2 First Results using DEAP

A first attempt to build a border model with Genetic Programming was conducted using the DEAP library (Distributed Evolutionary Algorithms in Python (Fortin et al., 2012)), a free open-source evolutionary computation framework that includes multiple features for quick prototyping and testing. The operators and terminals (two variables $X1$ and $X2$ in the two-dimensional case) were set as described above.

A first trial with DEAP was run on the two-dimensional test case #1 where the *NG* area is located in the lower right corner (see Figure 6.8). After a few trial-and-errors, for the 50 fitness cases of this use case, the best results were obtained with a population size of 1,000, a number of generations of 100 and a tournament size of 50. Although the best errors achieved were less than 10%, with very complicated analytical equations to represent the border, we decided to reach out to professionals with expert knowledge in the field to guarantee a correct implementation of Genetic Programming, and give us an idea on its potential as possible border model.

We contacted **MyDataModels**, a startup company specialized in self-service machine learning for small data, whose technology is based on a proprietary Genetic Programming engine called *TADA*. They offer a whole online platform on which GP models can easily be built from tabular data even by non-expert users. And the models generated by TADA are generally rather small.

6.4.3 Results using TADA

First, we tried the TADA model on test case #1 (see Figure 6.8) where a single NG area is present in the right lower corner. It was tested on initial sets of 50, 200 and 500 initial scenarios respectively. The numbers of generations and population size were kept to their default values (100 and 500 respectively). The test accuracies obtained are: 80%, 93.33% and 99.33% for the 50, 200 and 500 scenarios initial sets respectively. Furthermore, the platform provides a Python 3 script of two output functions, which are then compared to compute a prediction function that is able to compute if a given scenario is labelled *G* or *NG*. We thus used this function to predict the G/NG evaluations of a full grid of this test case. The results are shown in Figure 6.25.

We can observe that the resulting test accuracies reflect the capacity of the algorithm to precisely detect the border depending on the initial sets provided. Obviously, as could be expected, the bigger the initial set, the more information the algorithm has to precisely detect a finer border, the better the results.

Then, we tried the TADA model on the 10,000 set size extracted of the six-dimensional NHTSA 13 use case. We chose what variable corresponds to our goal in the study. Because we want a border model, we picked the output in our data set that gives us the information about whether the scenarios are "on" the border or not. Since it is a Boolean variable, the model directly considers the problem to be a classification task. Then, the input variables

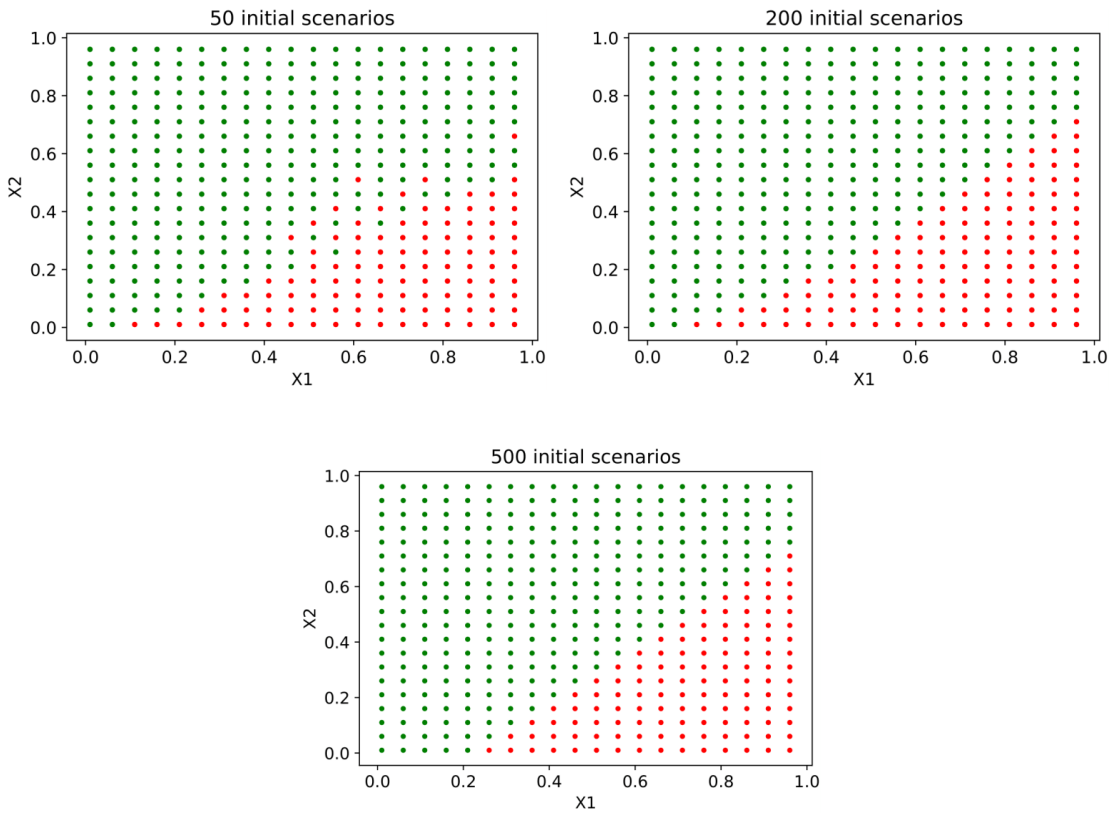

Figure 6.25: Visual representation of the TADA approach on a grid of test case #1 with a single *NG* corner area applied to 50, 200 and 500 initial scenarios.

which are allowed to be used by the model are defined. Next, the number of generations and the population size are defined. We chose the default parameters already picked: 100 generations and 500 population individuals. Finally, the model is launched by splitting the data set automatically into three data sets: a training set of 4,000 scenarios, a validation set of 3,000 scenarios and a testing set of 3,000 scenarios, and takes approximately 5 minutes to complete its operation which is deemed acceptable according to industrial restrictions. After it has finished computing, an overview of the model performance and results is accessible. For instance, it shows which variables it used in its tree representation, as well as test metric scores such as the accuracy. This border model achieved a test accuracy of 86.17% while only using three input variables out of the six: Initial velocity of EGO, Initial velocity of V1, and TTC threshold between V1 and V2 (see Table 6.1).

Next, we applied this model on the full grid of 470,587 scenarios. We obtained a total error of 14.2%: 13.4% for the border scenarios and 14.5% for the remaining scenarios. We decided then to try to perform some iterations while adding undecided scenarios, similarly as we have previously done with the Neural Network model in Section 6.2.2. Fortunately,

although the learned border model is here a classifier, with Boolean outputs, it also outputs a "confidence" in $[0, 1]$ that represents how much the model is certain about predicting the class for each scenario: in fact, the output of the tree is a real value, that gets rounded to 0 or 1, and the confidence is proportional to the proximity to 0 or 1. To have a better idea about this value, Figure 6.26 shows a histogram representing the cumulative density probability of that value.

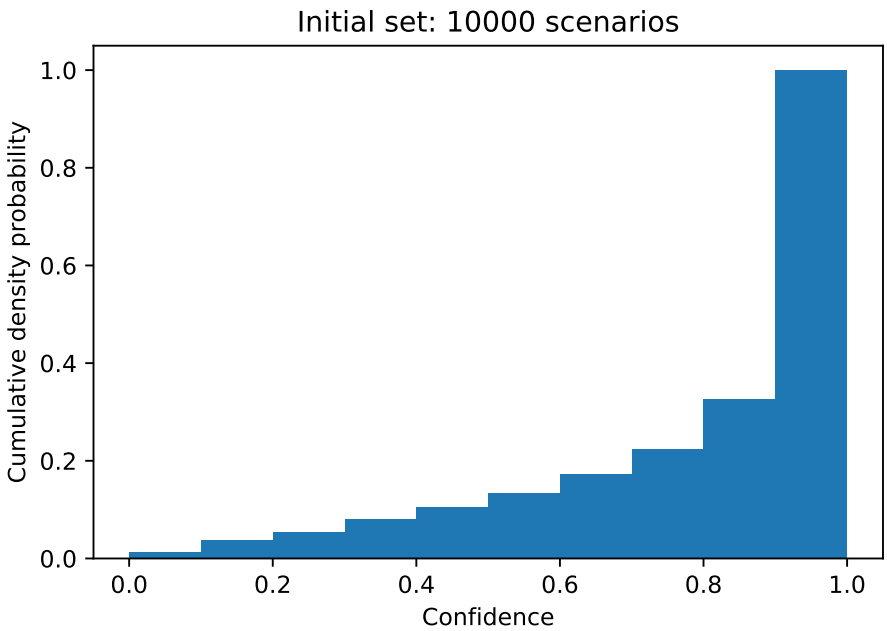

Figure 6.26: Histogram showing the cumulative density probability of the value "confidence" generated by the TADA model on the set of size 10,000 stemming from the use case NHTSA 13.

We can see in this histogram that approx. 70% of the scenarios have a confidence between 0.9 and 1, while the others output confidence values scattered between 0 and 0.9. The scenarios that have the lowest confidence value (between 0 and 0.1) are definitely the most undecided scenarios. Thus, we picked some of these and added them to the initial training set, then learnt a new TADA model, possibly iterating this process several times. Note that because TADA is an online platform, such iterations cannot be automatically performed, and all the steps have to be iterated manually. Furthermore, some limitations on the numbers of models and predictions allowed are imposed daily, which limited the scope of this experimentation overall. Nevertheless, we tested this model across 10 iterations to try to evaluate the model performance and the error values across the iterations. After completing these iterations, Figures 6.27 and 6.28 show the variation of the accuracy and the total error computed along the iterations.

We can see in Figure 6.27 that, similarly to the Neural Network context, the addition of undecided scenarios to the learning set does not imply an improved accuracy, at least

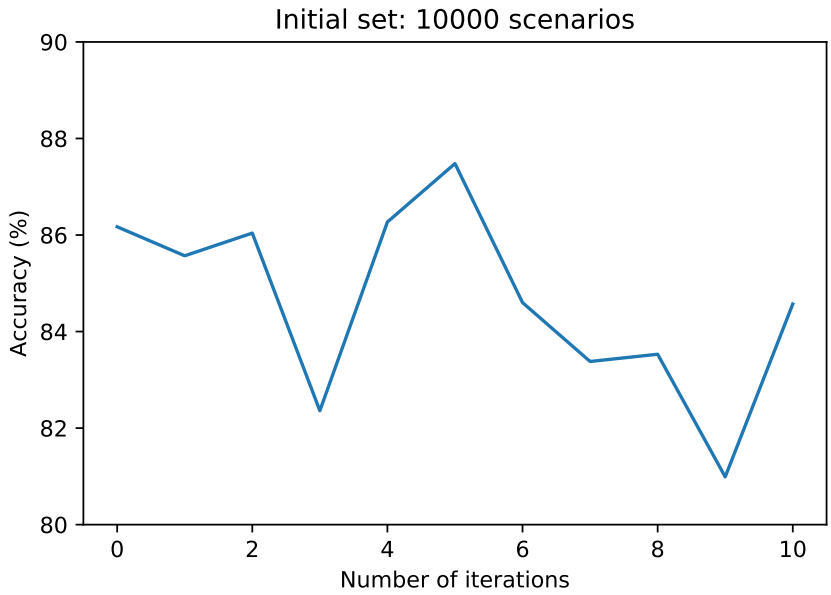

Figure 6.27: Variation of the accuracy of the TADA model on an initial set of 10,000 scenarios (5,000 border scenarios and 5,000 other scenarios) across the iterations of adding lowest confidence scenarios to the initial set.

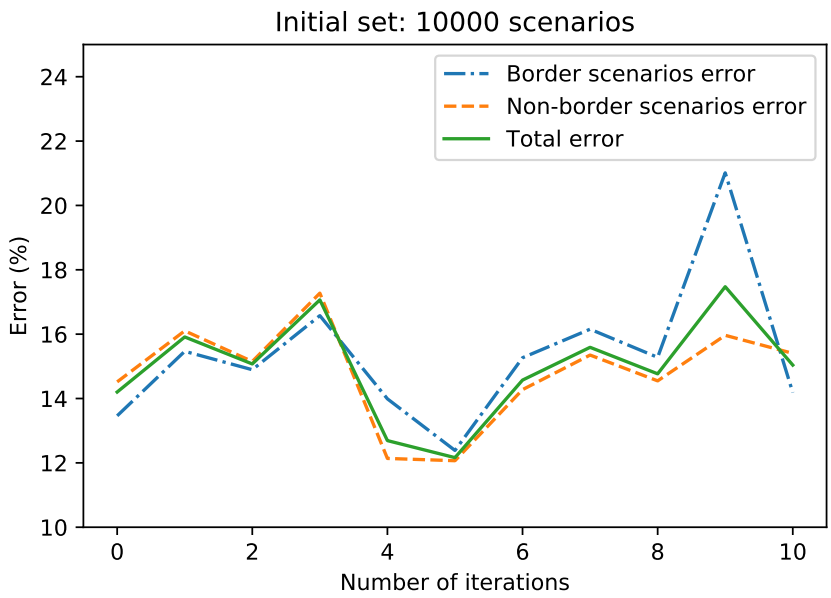

Figure 6.28: Variation of the total error computed on all 470,587 scenarios of the NHTSA 13 use case after applying the TADA model based an initial set of 10,000 scenarios (5,000 border scenarios and 5,000 other scenarios) across the iterations of adding lowest confidence scenarios to the initial set.

for the 10 first iterations. The initial accuracy was 86.17%, and during the iterations, it fluctuates between 81% and 88%. As for the errors computed on the full grid (see Figure 6.28), their values and trends mirror those obtained with the accuracy: The error increases when the accuracy decreases, and the total error takes values between 12% and 18%. While these graphs draw a lot of similarities seen in the results of the Neural Network border model applications, the values obtained by the Neural Network in accuracy and error are significantly better. However, the advantage of the evolutionary approach is the analytical form of the learned model, that gives us the clear equation of the border that was intended, similarly to the MILP approach in this respect.

In order to check the sensitivity of the results w.r.t. the hyper-parameters "number of generation" and "population size", we ran a series of single run experiments on different training sets while performing a grid search. The results (test accuracies) are presented in Table 6.5.

Table 6.5: Test accuracies (in %) of different models using TADA while changing the population size, number of generations and the initial set size.

Initial set size	**Population size**	**Number of generations**		
		100	500	1000
500	500	75.33	79.33	80.67
	1000	81.33	85.33	81.33
	5000	87.33	86	85.33
	10000	82	84.67	86
1000	500	81	85.33	85.67
	1000	85.67	83.33	83
	5000	83	89	90.33
	10000	88.67	88.33	88
2500	500	85.47	87.6	86.8
	1000	84.4	85.33	82
	5000	85.2	85.6	91.33
	10000	90	89.47	86.13
5000	500	83.2	87.07	89.4
	1000	81.53	88	85.53
	5000	84.8	87.27	86.07
	10000	89.2	91.6	88.53
10000	500	86.17	90.07	85.93
	1000	86.2	85.2	89.07
	5000	86.17	88.23	88.17
	10000	85.6	88.97	92.1

Table 6.5 shows no clear link between the variation of the accuracy and the increase of the number of generations and population size parameters. All accuracy values are found between 75.33% (Initial set: 500, Population size: 500, Number of generations: 100) and

92.1% (Initial set: 10000, Population size: 10000, Number of generations: 1000), which are inferior to the values obtained with the previous border models.

6.5 Discussion

In short, all three approaches have proven their value in addressing the problem and embodying an acceptable border model.

1. The **Neural Network** represents the problem well after being applied on several test cases. It achieves accuracy values higher than 95% on the test sets with acceptable error classification values around 5%. The disadvantages of the Neural Network is the possibility of being stuck in local minima, and its inability in expressing a clear equation that represents the border as it is required for this final thesis objective.

2. The **MILP** approach went into different editions leading to a successful industrialization. This low-cost model gives a clear tree representation and is mainly oriented into mathematical programming. Its gives clear final equations of the border and delivers classifications errors lower than the Neural Network. Since the Greedy edition got recently industrialized, it should be tested on a lot of different test cases and be enhanced and updated continuously to make sure that it broadens its application range on all possible use cases.

3. Lastly, the **evolutionary** approach proved to be a promising direction to pursue the research in the border model area. It can be seen as middle ground between the two previous approaches, as it can give a clear tree representation at the end while being easily applied to any type of test case. Nonetheless, the only current hiccup of the evolutionary approach is the performance of the model as seen in the accuracy and error values obtained. While the approach was tested on a small set of experiments only, more research should be conducted to try to improve these values and evaluate more thoroughly the potential of the evolutionary border model.

Chapter 7

Conclusions

Building a fully autonomous vehicle seems like the natural evolution of automation in automotive. After perfecting mechanical and electrical systems within the vehicle, the interest is shifted toward electronic systems dubbed Advanced Driver-Assistance Systems (ADAS) that aim to assist the vehicle driver while driving and parking. The combination of all these sophisticated ADAS will lead to the creation of the fully autonomous vehicle. The ultimate goal is, obviously, to reduce car accidents and increase the overall safety on the roads. Multiple other benefits underpin the phenomenon of building such vehicle by all major Original Equipment Manufacturers (OEMs) such as Renault, as well as joined by recent technology development companies. These advantages include environmental benefits, more independence and access to driving to all types of people, and overall a better health for all passengers on board.

However, when conceiving such vehicle, all companies quickly discovered the growing complexity of a fully autonomous vehicle. Categorizing the vehicle autonomy into levels, from no automation (level 0) to full automation (level 5) by SAE (2018), seemed the most reasonable solution to tackle this challenge one step at a time. Each level is defined by a set of specific conditions under which a given driving automation system is designed to function, otherwise called Operational Design Domain (ODD), such as traffic conditions, geographic location, speed range etc. Then, the idea is to complete the technical hurdles of each level before going to the next while updating all systems accordingly. Naturally, a higher level is defined by more autonomy features, which in turn requires more different sensors and more complex algorithms. And in order to validate a level, one must make sure that no system failures could occur, which could lead to some undesirable car malfunctions. Therefore, the main challenge of an autonomous vehicle is to test and validate thoroughly every system and component before industrialization, in order to mitigate potential failures and avoid unwanted problems on the road.

In an autonomous vehicle, four main systems are present. First, the perception system is composed of all the sensors needed to detect outside elements from the car environment. Then, the fusion system receives the objects detected and synchronizes them, before merging

the information into a unique and more precise mapping of the vehicle environment. Next, the decision system is fed with all final objects, and computes what the vehicle should be doing on the road, based on the information transmitted by the fusion system, following a predefined command law. Finally, the actuators perform all requested instructions prescribed by the decision system. Hence, the autonomous vehicle represents a highly complex system, and a failure can occur during any stage of this process. These failures include perception and fusion system errors, misinterpretation by the decision system, and actuator latency. Plus, it can sometimes be impossible to trace which component is responsible for a given failure, due to the complexity of the system as a whole. This is why multiple validation techniques are explored to form an extensive testing and validation required for each component.

Naturally, software testing is conducted on the million lines of codes included in the algorithms of all the different components. Nonetheless, this remains insufficient in ensuring quality testing, as the validation of autonomous vehicles faces numerous challenges (Koopman and Wagner, 2018). Two main categories of testing are present nowadays. First, real test-driving is used by major car companies to test the vehicle under various conditions and detect specific system failures. They can rely on public road testing, though country regulations do not allow such tests everywhere, and closed course testing, which is safer and less costly, but not scalable to all possible driving situations on public roads. However, this approach cannot be solely used to complete the validation process, as studies have shown that hundreds of years of continuous real test-driving would be needed to demonstrate with 95% confidence that the autonomous vehicle failure rate is lower than that of the human driver (Kalra and Paddock, 2016).

That is why, thanks to the increase in computing power nowadays, simulation testing methods are used to complement real test-driving; And they are also way cheaper. Three main subcategories can be distinguished here: resimulation injects real driving data into a numerical model of the command law to try to replicate them in an open loop (while closed-loop resimulation remains a research and development subject), numerical simulation generates virtual parameterized data and creates new tests by relying on models of all the vehicle systems, and the virtual reality simulator allows the person to drive a real vehicle while connected to its environment through a simulator software. All these techniques are used in the V-model applied to the validation of the autonomous vehicle, which is designed to test and validate step by step all requirements until the final acceptance test.

The context of this PhD thesis is that of the numerical simulation testing approach. People at Renault have designed a complete simulation-based validation chain in order to address all possible driving situations, also called use cases, which are defined by inputs that characterize the parameters influencing the autonomous vehicle environment. A combination of these inputs defines a scenario, which can be launched into an in-house simulation software called SCANeR Studio (AVSimulation, 2017). Plus, within this validation chain, a massive simulation platform is being built to accommodate the needs of launching multiple scenarios at the same time, which is useful in validating the decision system requirements by performing

Model-In-the-Loop (MIL) testing. MIL testing refers to a simulated environment of the model to be tested in the absence of physical hardware components. That way, we can verify that the actions chosen by the vehicle remain safe according to the driving rules, and numerically validate the decision system requirements while exploring a multitude of scenarios for a defined use case.

Therefore, the idea of this PhD thesis is to facilitate the validation of the decision system by aiding the MIL simulations in exploring the use case input space. It takes part in an industrial project at Renault called ADValue (Tourbier, 2017), which aims at combining multiple algorithms that compete in exploring the use case input space efficiently to cover all failure zones in an intelligent manner. In order to do that, the ultimate goal is to conceive models and develop algorithms that use time and available resources efficiently, in order to avoid simulating all imaginable conditions of all use cases, which would be highly expensive in terms of computing power. The aim of this thesis is then to feed the industrial project with novel algorithms and methods to achieve its objective.

The main contributions of this PhD thesis are threefold:

1. **Failure detection**

   One algorithm is developed to detect a maximum number of failures of the autonomous vehicle command law while exploring the input search space for a given use case. In order to abide with the industrial requirement of reducing the overall computing power needed, which translates into using the simulator as little as possible, a Random Forest model is built as a surrogate model of the simulator, and is used intensively through the optimization loop (Nabhan et al., 2019).

   When compared with the current state of the art review around this objective, the novelty brought by this algorithm is that it tackles coverage and falsification simultaneously to identify a maximum number of failures everywhere in the search space. Most of the publications focus on coverage or falsification alone (Ben Abdessalem et al., 2016; Beglerovic et al., 2017) by using Neural Networks and Kriging models as surrogate models, while some others try to combine the two objectives but to a different aim than of this thesis, e.g., to improve the optimizer convergence by enhancing its initialization through the selected scenarios (Tuncali et al., 2020).

2. **Border detection**

   Three other algorithms are developed to detect scenarios as near as possible from the border between faulty and non-faulty areas. In fact, use cases are usually defined by inputs of continuous nature, such as velocities, accelerations and decelerations of traffic vehicles surrounding the autonomous vehicle, and the use case input space can then be seen as a partition in different zones of failure-prone and failure-free scenarios. Each of the three proposed algorithms tackles the objective in a different way, while employing the same general strategy used by the failure detection algorithm to reduce computing power.

One single publication was found in the literature review that determines boundary scenarios witnessing the transition from safe driving to collision occurrences (Tuncali and Fainekos, 2019). They use Rapidly exploring Random Trees (RTTs) to define a custom function based on collision-related parameters, e.g., collision surface and velocity. The main difference with the work in this thesis is that the three algorithms are developed to detect the border of any evaluation criterion wanted, and not just for collision-related criteria, without the need of generating trajectories for all traffic vehicles.

3. **Border models**

   For the third part of this work, three approaches are considered to analytically identify the border itself as accurately as possible through direct or parameterized equations: Neural Networks, Mixed-Integer Linear Programming (MILP) with extensions, and Genetic Programming (GP) for symbolic regression. They are each built using scenarios that are known to be on, or very close to, the border, and could have been the results of the Border Detection algorithms – though this was not the case here. No relevant research was found in the literature review that addresses this objective.

The algorithms related to the first two objectives (that all identify scenarios, based on diverse criteria) are tested on a tracking vehicle use case, and their results are compared to a full grid of the input search space, with various metrics used to evaluate the quality of the results obtained for the border detection objective. The models related to the third objective are tested on a more complex use case and benefited from an updated simulator. They are compared by examining their total misclassification errors. All results obtained with the proposed algorithms are discussed in the next section.

7.1 Discussions

Failure detection For this first objective, we began by developing an algorithm that aims at identifying a maximum number of failures of the autonomous vehicle command law for a given use case. The algorithm then goes through an optimization loop to meet this objective.

Surrogate models In order to cope with the industrial project restrictions, we use a reduced model of the simulator intensively in the optimization loop. This Random Forest model is continuously updated throughout the iterations by checking the generated scenarios by actual simulations. Furthermore, technical limitations of the simulator did not enable the launch of a massive simulation plan in parallel mode, which could otherwise be helpful in developing and testing the algorithm. Therefore, a Neural Network serves as a substitution model of the simulator in this work. It is fed offline with calculations of the simulation software, and is never updated once built (considered as the simulation ground truth). The choices for the 'surrogate' models are justified by their characteristics and usage for this optimization task, as the reduced model, for instance, is needed to be fast to train, accurate

and robust against overfitting, because it is being updated at each iteration: this lead to the choice of Random Forest. Plus, we noticed that the reduced model is more prone to wrongly classify the scenarios generated during the first iterations, since it is only based on a handful number of scenarios, but quickly gains in accuracy thanks to the continuous update throughout the iterations (see Figure 4.4). As for the Neural Network, it is important to note that it should be replaced by the actual simulator as soon as the massive simulation platform is ready to be used. Until then, we made sure to choose the most convenient architecture and tune carefully all hyper-parameters to have a substitution model as accurate as possible, while being aware that a modeling error is being introduced to the results, especially if the simulator is being updated continuously. We can then consider that the Neural Network represents a lesser version of the simulator, until linking the simulator directly becomes a feasible option.

Optimization loop The algorithm **Find All Failures** aims to detect a maximum number of failures anywhere in the scenario space. For this objective, it repeatedly runs an instance of another algorithm dubbed **Find One Failure** whose role is to detect one failed scenario (labelled as NG) as far as possible from the known NG scenarios, already available in an archive of already simulated scenarios. **Find One Failure** uses CMA-ES (Hansen and Ostermeier, 2001) as optimization engine, optimizing an objective function that purposefully penalizes G scenarios in order to push the algorithm into detecting NG scenarios as far as possible from the dynamic archive. The Random Forest model is then intensively used by the optimization engine to get access to the scenarios warnings without the need of running the actual simulation. Note that there were other candidate algorithms for such a derivative-free optimization, such as Particle Swarm Optimization (PSO) or Differential Evolution (DE). However, on the one hand, it is known that CMA-ES outperforms PSO and DE on the whole set of BBOB benchmark functions (Hansen et al., 2011), and comparing these algorithms with one another was not an objective of this thesis. On the other hand, this first objective of this thesis has been already tackled recently in the literature, and several works use PSO and DE, for instance in falsification-based models (Beglerovic et al., 2017). This is why we decided to choose CMA-ES for its known global optimization performances, and devote our time to the more ambitious border-related objectives, which are rather yet unrepresented in the field.

Minimal precision distance The **Find All Failures** algorithm has been tested on a tracking vehicle use case, while fixing a minimal precision distance to be respected in the input search space. In that way, the algorithm is forced to explore the whole search space for failed scenarios. Then, its performance is assessed by comparing the results obtained to a full discretized grid of the search space. Discovery rates are computed, which represent the number of times the algorithm managed to predict a NG scenario close enough to each NG grid scenario, according to the minimal precision distance fixed. While results show

that the algorithm manages to attain high discovery rates, it is crucial to note that the minimal precision distance value greatly influences the number of failures obtained, which in turn impacts the number of simulations needed to cover the whole search space. Hence, a trade-off has to be made between detecting a maximum number of failures accurately, and launching as few software simulations as possible. However, due to time constraints and technical limitations of the simulator during most of the duration of this thesis, playing around with this minimal precision distance was not possible, and is left for future work.

Border detection Border detection is a more ambitious objective: it considers the input search space as a partition of areas of G/NG scenarios, and tries to detect scenarios near the border between these areas. In fact, the first failure detection algorithm could be inefficient for all the stages of the industrial project ADValue, especially towards the end where the NG scenarios become rarer to identify since the autonomous vehicle command law is theoretically being updated and enhanced throughout the project stages. Therefore, developing other algorithms that aim at detecting directly the border, would be beneficial for the ADValue project to anticipate the possible coming stages, by proposing multiple algorithms that work together in the validation process whatever the stage at hand. Furthermore, an important criterion for the domain expert lies in understanding the scenario space, in order to be able to propose corrections of the command law that apply to whole areas and not only to single points of failure as they are identified. For this purpose, three algorithms with different approaches have been developed: **Find Border Points** identifies on-the-border scenarios, while **Find Border Max** and **Find Border Min** generate close G/NG pairs between which the border should precisely be located.

Stopping condition While the stopping condition elaborated for the failure detection algorithm is related to the minimal precision distance in the search space, choosing a stopping condition for the border detection algorithms turned out to be more challenging. Because each algorithm functions differently (by identifying G/NG pairs or on-the-border scenarios), setting a common stopping condition for the sake of comparison could only be achieved by fixing the number of simulations (or number of Neural Network calls in this work) as stopping condition, which is compatible with the industrial project restrictions of reducing computing power needed. Note that in the ADValue project, all algorithms would be given the same stopping condition, but to a much lesser number of simulations. In this work, we conducted experiments for 1,000 and 3,000 simulations, whereas the algorithms in the ADValue project are expected each to propose a handful of scenarios, and the results are either compared or merged depending on the overall efficiency score obtained. By choosing 1,000 and 3,000 simulations, we are testing the full potential of each algorithm separately, in order to obtain a clear evidence of what one can expect from these algorithms when tackling this challenging border-related goal.

Qualitative results At first, we tried evaluating the results quality with precision and border rates while comparing to a full grid over the search space. On one hand, the precision rate metric is computed as the ratio of all the occurrences of equal criteria evaluations between the grid scenarios and their nearest algorithm-generated neighbors, to the total number of grid scenarios. On the other hand, the border rate metric is the ratio of the G/NG grid couples whose warning evaluations remained unchanged after modifying their evaluations according to their nearest algorithm neighbors warnings, to all the G/NG grid couples identified before any algorithm action. Then, connected components were eventually considered because they overcome the limitations of the first two metrics: precision rates were unable to give a clear representation of the performances, and border rates are limited in taking into account a possible offset in the border detection. Besides, connected components give access to interesting evaluation parameters, namely the "offset" distances between the scenarios incorrectly classified as NG by the algorithms and the original connected components (to evaluate the offset prediction error), and the "coverage" distances between the scenarios correctly classified as NG by the algorithms and the NG scenarios of the original connected components that remained undetected by the algorithms (to assess the coverage of the algorithms). While the results show that all three algorithms are able to minimize the possible offset in the border detection throughout the iterations, **Find Border Points** manages to consistently improve throughout the simulations and effectively explore the NG areas for all warnings, whereas both other algorithms generate acceptable or mediocre results depending on the border criterion under evaluation. This is explained due to the fact that **Find Border Points** is conceived in a way that allows the algorithm to explore everywhere in the search space, whereas the **Max** and **Min** algorithms are both limited by their initial set and cannot explore outside of their convex hulls. Nevertheless, they can be useful when injecting them to the ADValue project by focusing on some areas of interest and targeting the border in specific locations, while **Find Border Points** explores the search space for new unidentified border locations.

Border models After having developed algorithms capable of identifying border scenarios for a given use case, the last objective of this thesis aims at building analytical and hence hopefully explainable border models, which can identify the border itself as accurately as possible through direct or parameterized equations while providing insights on the shape of the accuracy in its neighborhood. In fact, the border detection algorithms of Chapter 5 give a general, but rather sparse, insight into the possible location of the border in the search space. The border models, on the opposite, while possibly based on border scenarios provided by these algorithms, try to define globally the border more precisely. Three different approaches have been tested and compared: Neural Networks, Mixed-Integer Linear Programming (MILP) with extensions, and Genetic Programming (GP) for symbolic regression.

Models comparison First, the Neural Network approach is considered, and is tested on several test cases, including a more sophisticated use case called NHTSA 13 (see Figure 6.1). After managing to achieve accuracy values higher than 95% for small data sets, classification errors are computed based on predictions of a full discretization of the search space. The results show acceptable errors around 5% depending on the data set size, but present disadvantages of this approach, which are the possibility of being stuck in local minima. The lack of explainability of the trained model is another drawback of this approach. Nonetheless, the Neural Network is considered as a reference for the other approaches in terms of accuracy. Next, since we are interested in clear equations that describe the border, we considered different approaches based on MILP, namely the Global algorithm (finding all border axes simultaneously), the Iterative algorithm (detecting one border axis at a time and calling upon a clustering algorithm when blocked) and the Greedy algorithm (same as One Axis but limits the use of clustering within the generation process). The goal is to produce a low-cost mathematical programming model, that gives a clear tree representation for the given use case border, as well as the equations that delineate the G and NG areas. The One Axis model obtained classification errors even lower than that of the Neural Network, and the final Greedy edition was successfully industrialized by EURODECISION, and should be tested intensively and updated continuously within the ADValue project to ensure a broad application range on all possible use cases. Finally, we explored the evolutionary approach by applying Genetic Programming to this symbolic regression task. Evolutionary Algorithms are known for their wide application range, and Genetic Programming also evolve tree representations of the solution; this approach can be seen as intermediate between both previous methods. However, although the tests realized on the commercial MyDataModels online platform show acceptable accuracies above 80%, their performances were repeatedly less better than the ones obtained with both previous methods. Nonetheless, this approach could be considered by Renault as a potential future direction of research.

7.2 Perspectives

Since the ADValue project consists of mixing multiple algorithms working in competition to find the best scenarios within the validation process, every novel algorithm developed can be integrated into the project. For instance, a promising extension of the **Find Border Points** algorithm could be explored: Since the algorithm uses CMA-ES as optimization engine for the detection of border scenarios, inspiration can be drawn from the failures detection algorithm in the way it was conceived. In fact, the **Find All Failures** algorithm searches for the NG scenarios the farthest possible from existing NG scenarios in the archive, using intensively CMA-ES to propose such scenarios. The result is an effective exploration of the whole search space for failed scenarios. The same methodology could be incorporated into **Find Border Points** to create a new algorithm that searches for border scenarios the farthest possible from existing border scenarios. Since we now have three algorithms that

generate such scenarios, this new algorithm could arrive in a later phase of the project, when border scenarios are already available in the search space, and can also attempt at a complete and effective coverage of border scenarios, which will undoubtedly also aid the border models generated afterwards.

Furthermore, since the Greedy MILP edition was successfully industrialized, testing should be conducted on various use cases to evaluate its overall performance, its scalability, and delineate its application ranges. It is being considered to be used as stopping condition for the ADValue project; after sufficient iterations and generation of NG and border scenarios by all algorithms, the border model is built and computes the total misclassification error. If given a certain error goal, the project resumes until the error value is reached for the given use case, which indicates a successful assimilation of the border for that use case. Naturally, a lot of research remains to be conducted. In particular, we merely tackled small use cases of dimensions 5 and 6 in this work, which are considered simple use cases compared to bigger and much more complicated use cases with tens of input variables. Thus, operational application of these models and algorithms can only become a reality after demonstrating that they actually can scale up to much larger use case dimension, eventually through incremental progress and validating each use case one at a time, in order to guarantee a good conclusion for the ADValue project, and a successful scenario-based validation for the autonomous vehicle command law at large.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Abbas, H., O'Kelly, M., Rodionova, A., and Mangharam, R. (2017). Safe at any speed: A simulation-based test harness for autonomous vehicles. In *7th Workshop on Design, Modeling and Evaluation of Cyber Physical Systems (CyPhy17)*.

Akagi, Y., Kato, R., Kitajima, S., Antona-Makoshi, J., and Uchida, N. (2019). A risk-index based sampling method to generate scenarios for the evaluation of automated driving vehicle safety*. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 667–672.

Ali, J., Khan, R., Ahmad, N., and Maqsood, I. (2012). Random forests and decision trees. *International Journal of Computer Science Issues(IJCSI)*, 9.

Althoff, M. and Dolan, J. M. (2012). Reachability computation of low-order models for the safety verification of high-order road vehicle models. In *2012 American Control Conference (ACC)*, pages 3559–3566.

Althoff, M. and Lutz, S. (2018). Automatic generation of safety-critical test scenarios for collision avoidance of road vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1326–1333.

Angeline, P. and Pollack, J. (1997). Evolutionary module acquisition. *Proceedings of the Second Annual Conference on Evolutionary Programming*.

Arief, M., Glynn, P., and Zhao, D. (2018). An accelerated approach to safely and efficiently test pre-production autonomous vehicles on public streets. *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2006–2011.

Auger, A. and Hansen, N. (2005). A restart CMA evolution strategy with increasing population size. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005, 2-4 September 2005, Edinburgh, UK*, pages 1769–1776. IEEE.

Auger, A., Hansen, N., Perez Zerpa, J. M., Ros, R., and Schoenauer, M. (2009). Experimental Comparisons of Derivative Free Optimization Algorithms. In Vahrenhold, J., editor, *8th International Symposium on Experimental Algorithms*, LNCS, Dortmund, Germany. Springer Verlag.

AVSimulation (2017). SCANeR Studio. https://www.avsimulation.com/solutions/.

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998). *Genetic Programming - An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann.

Beglerovic, H., Ravi, A., Wikstroem, N., Koegeler, H.-M., Leitner, A., and Holzinger, J. (2017). Model-based safety validation of the automated driving function highway pilot. In *8th International Munich Chassis Symposium 2017 chassis.tech plus*. Springer.

Beglerovic, H., Stolz, M., and Horn, M. (2017). Testing of autonomous vehicles using surrogate models and stochastic optimization. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6.

Belbachir, A., Smal, J. C., Blosseville, J. M., and Gruyer, D. (2012). Simulation-driven validation of advanced driving-assistance systems. In *Procedia-Social and Behavioral Sciences*, volume 48, pages 1205–1214.

Ben Abdessalem, R., Nejati, S., Briand, L. C., and Stifter, T. (2016). Testing advanced driver assistance systems using multi-objective search and neural networks. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 63–74.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305.

Berkelaar, M., Eikland, K., and Notebaert, P. (2004). lp_solve 5.5, open source (mixed-integer) linear programming system. Software. Available at ¡http://lpsolve.sourceforge.net/5.5/¿. Last accessed Dec, 18 2009.

Biau, G. (2012). Analysis of a random forests model. *J. Mach. Learn. Res.*, 13(1):1063–1095.

Brameier, M. F. and Banzhaf, W. (2010). *Linear Genetic Programming*. Springer Publishing Company, Incorporated, 1st edition.

Breiman, L. (1996). Bagging predictors. *Mach. Learn.*, 24(2):123–140.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA.

Breyer, G. (2010). Safe distance between vehicles. Conference of European Directors of Roads (CEDR).

Chen, X., Wang, K., and Yang, F. (2013). Stochastic kriging with qualitative factors. In *Proceedings of the 2013 Winter Simulation Conference*, pages 790–801. IEEE.

Connor, M., Fagan, D., and O'Neill, M. (2019). Optimising team sport training plans with grammatical evolution. In *IEEE Congress on Evolutionary Computation, CEC 2019, Wellington, New Zealand, June 10-13, 2019*, pages 2474–2481. IEEE.

Corso, A., Du, P., Driggs-Campbell, K. R., and Kochenderfer, M. J. (2019). Adaptive stress testing with reward augmentation for autonomous vehicle validatio. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 163–168.

Douglas, J. W. and Schafer, T. C. (1971). The chrysler "sure-brake" - the first production four-wheel anti-skid system. In *SAE Technical Paper*. SAE International.

Echard, B., Gayton, N., and Lemaire, M. (2011). AK-MCS: An active learning reliability method combining kriging and monte carlo simulation. *Structural Safety*, 33(2):145–154.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21.

Fahrenkrog, F., Wang, L., Platzer, T., and Raisch, F. (2019). Prospective safety effectiveness assessment of automated driving functions – from the method to the results. In *26th International Technical Conference on the Enhanced Safety of Vehicles (ESV)*.

Feig, Philip, Schatz, Julián, and Labenski (2019). Assessment of technical requirements for level 3 and beyond automated driving systems based on accident data analysis. In *26th International Technical Conference on the Enhanced Safety of Vehicles (ESV)*.

Felbinger, H., Klück, F., Li, Y., Nica, M., Tao, J., Wotawa, F., and Zimmermann, M. (2019). Comparing two systematic approaches for testing automated driving functions. In *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*, pages 1–6.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc.

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., and Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175.

FWHWA and US Department of Transportation (2017). NGSIM-Next Generation SIMulation. https://ops.fhwa.dot.gov/trafficanalysistools/ngsim.htm.

Gangopadhyay, B., Khastgir, S., Dey, S., Dasgupta, P., Montana, G., and Jennings, P. (2019). Identification of test cases for automated driving systems using bayesian optimization. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 1961–1967.

Gao, F., Duan, J., He, Y., and Wang, Z. (2019). A test scenario automatic generation strategy for intelligent driving systems. *Mathematical Problems in Engineering*, 2019:1–10.

Gass, S. (2002). The first linear-programming shoppe. *Operations Research*, 50:61–68.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition.

Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., and Sculley, D. (2017). Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495. ACM.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Neural Information Processing Systems (NIPS)*, pages 2672–2680.

Gordon, A., Eban, E., Nachum, O., Chen, B., Wu, H., Yang, T.-J., and Choi, E. (2018). Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1586–1595.

Guo, J., Kurup, U., and Shah, M. (2019). Is it safe to drive? an overview of factors, metrics, and datasets for driveability assessment in autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, PP:1–17.

Gurobi Optimization, L. (2020). Gurobi optimizer reference manual.

Hansen, N., Finck, S., and Ros, R. (2011). COCO - COmparing Continuous Optimizers : The Documentation. Research Report RT-0409, INRIA.

Hansen, N. and Kern, S. (2004). Evaluating the CMA evolution strategy on multimodal test functions. In Xin Yao et al., editor, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science*, pages 282–291. Springer.

Hansen, N., Müller, S., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11:1–18.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Hebb, D. O. (1949). *The organization of behavior: A neuropsychological theory.* Wiley, New York.

Huang, L., Xia, Q., Xie, F., Xiu, H.-L., and Shu, H. (2018). Study on the test scenarios of level 2 automated vehicles. *Intelligent Vehicles Symposium*, pages 49–54.

Huang, W., Wang, K., Lv, Y., and Zhu, F. (2016). Autonomous vehicles testing review. In *IEEE 19th International Conference on Intelligent Transportation Systems (ITSC).*

Huang, Z., Arief, M., Lam, H., and Zhao, D. (2019). Evaluation uncertainty in data-driven self-driving testing. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 1902–1907.

Huang, Z., Lam, H., LeBlanc, D. J., and Zhao, D. (2018). Accelerated evaluation of automated vehicles using piecewise mixture models. *IEEE Transactions on Intelligent Transportation Systems*, 19(9):2845–2855.

Huang, Z., Lam, H., and Zhao, D. (2017). Sequential experimentation to efficiently test automated vehicles. In *Proceedings of the 2017 Winter Simulation Conference*, pages 3078–3089. IEEE.

Huang, Z., Lam, H., and Zhao, D. (2017a). Towards affordable on-track testing for autonomous vehicle — a kriging-based statistical approach. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6.

Huang, Z., Zhao, D., Lam, H., LeBlanc, D. J., and Peng, H. (2017b). Evaluation of automated vehicles in the frontal cut-in scenario — an enhanced approach using piecewise mixture models. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 197–202.

Ito, T., Iba, H., and Sato, S. (1998). Non-destructive depth-dependent crossover for genetic programming. In *Proceedings of the First European Workshop on Genetic Programming*, EuroGP '98, page 71–82, Berlin, Heidelberg. Springer-Verlag.

Jin, Y. (2011). Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61 – 70.

Johnson, M., Moore, L. M., and Ylvisaker, D. (1990). Minimax and maximin distance designs. *Journal of Statistical Planning and Inference*, 26(2):131–148.

Jones, D. R., Schonlau, M., and Welch, W. J. (2006). Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):4550492.

Junietz, P., Steininger, U., and Winner, H. (2019). Macroscopic safety requirements for highly automated driving. *Transportation Research Record*, 2673(3):1–10.

Junietz, P., Wachenfeld, W., Klonecki, K., and Winner, H. (2018). Evaluation of different approaches to address safety validation of automated drivings. In *21st IEEE International Conference on Intelligent Transportation Systems*, pages 491–496.

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

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming-ii. *Combinatorica*, 4:373–395.

Kennedy, J. and Eberhart, R. (1995). Particle Swarm Optimization. In *IEEE International Conference on Neural Networks. IV*, page 1942–1948. IEEE Press.

Khastgir, S., Dhadyalla, G., Birrell, S., Redmond, S., Addinall, R., and Jennings, P. (2017). Test scenario generation for driving simulators using constrained randomization technique. In *SAE Technical Paper*. SAE International.

Kim, B., Jarandikar, A., Shum, J., Shiraishi, S., and Yamaura, M. (2016). The smt-based automatic road network generation in vehicle simulation environment. In *Proceedings of the 13th International Conference on Embedded Software*, EMSOFT '16, New York, NY, USA. Association for Computing Machinery.

Kim, B., Masuda, T., and Shiraishi, S. (2019). Test specification and generation for connected and autonomous vehicle in virtual environments. *ACM Trans. Cyber-Phys. Syst.*, 4(1).

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Klischat, M. and Althoff, M. (2019). Generating critical test scenarios for automated vehicles with evolutionary algorithms. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2352–2358.

Koopman, P. and Fratrik, F. (2019). How many operational design domains, objects, and events? In *SafeAI@AAAI*.

Koopman, P. and Wagner, M. (2016). Challenges in autonomous vehicle testing and validation. *SAE Int. J. Trans. Safety*, 4(1):15–24.

Koopman, P. and Wagner, M. (2017). Autonomous vehicle safety: An interdisciplinary challenge. *IEEE Intelligent Transportation Systems Magazine*, 9(1):90–96.

Koopman, P. and Wagner, M. (2018). Toward a framework for highly automated vehicle safety validation. In *SAE Technical Paper*. SAE International.

Koren, M., Alsaif, S., Lee, R., and Kochenderfer, M. J. (2018). Adaptive stress testing for autonomous vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–7.

Koschi, M., Pek, C., Maierhofer, S., and Althoff, M. (2019). Computationally efficient safety falsification of adaptive cruise control systems. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2879–2886.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Koza, J. R. (1993). Hierarchical automatic function definition in genetic programming. In WHITLEY, L. D., editor, *Foundations of Genetic Algorithms*, volume 2 of *Foundations of Genetic Algorithms*, pages 297 – 318. Elsevier.

Krajewski, R., Bock, J., Kloeker, L., and Eckstein, L. (2018). The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2118–2125.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *NIPS*.

Lakomicki, P. (2018). *Incremental framework to qualify the reliability of the autonomous vehicle's perception and decision system*. PhD thesis, Université de technologie de Troyes, LM2S.

Lee, D. and Hess, D. J. (2020). Regulations for on-road testing of connected and automated vehicles: Assessing the potential for global safety harmonization. *Transportation Research Part A: Policy and Practice*, 136:85 – 98.

Lee, R., Kochenderfer, M. J., Mengshoel, O. J., Brat, G. P., and Owen, M. P. (2015). Adaptive stress testing of airborne collision avoidance systems. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, pages 6C2–1–6C2–13.

Lee, R., Mengshoel, O., Saksena, A., Gardner, R., Genin, D., Brush, J., and Kochenderfer, M. J. (2018). Differential adaptive stress testing of airborne collision avoidance systems. In *2018 AIAA Modeling and Simulation Technologies Conference*.

Majzik, I., Semeráth, O., Hajdu, C., Marussy, K., Szatmári, Z., Micskei, Z., Vörös, A., Babikian, A., and Varró, D. (2019). Towards system-level testing with coverage guarantees for autonomous vehicles. In *IEEE / ACM 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, IEEE. New Ideas and Vision Track.

Miller, J. F. (2020). Cartesian Genetic Programming: its status and future. *Genetic Programming and Evolvable Machines*, 21(1-2):129–168. Twentieth Anniversary Issue.

Mullins, G. E. (2018). *Adaptive Sampling Methods for Testing Autonomous Systems*. PhD thesis, University of Maryland, College Park.

Mullins, G. E., Stankiewicz, P. G., and Gupta, S. K. (2017). Automated generation of diverse and challenging scenarios for test and evaluation of autonomous vehicles. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1443–1450.

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

Nabhan, M., Schoenauer, M., Tourbier, Y., and Hage, H. (2019). Optimizing coverage of simulated driving scenarios for the autonomous vehicle. In *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*.

Olivares, S., Rebernik, N., Eichberger, A., and Stadlober, E. (2016). *Virtual Stochastic Testing of Advanced Driver Assistance Systems*, pages 25–35. Springer Verlag.

Orden, A. (1993). Lp from the '40s to the '90s. *INFORMS Journal on Applied Analytics*, 23(5):2–12.

O'Kelly, M., Sinha, A., Namkoong, H., Duchi, J., and Tedrake, R. (2018). Scalable end-to-end autonomous vehicle testing via rare-event simulation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 9849–9860, Red Hook, NY, USA. Curran Associates Inc.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau,

D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Perkis, T. (1994). Stack-based genetic programming. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 148–153 vol.1.

Pierson, A., Schwarting, W., Karaman, S., and Rus, D. (2019). Learning risk level set parameters from data sets for safer driving. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 273–280.

Poli, R., Kennedy, J., and Blackwell, T. (2007). Particle swarm optimization: An overview. *Swarm Intelligence*, 1.

Ponn., T., Fratzke., D., Gnandt., C., and Lienkamp., M. (2019). Towards certification of autonomous driving: Systematic test case generation for a comprehensive but economically-feasible assessment of lane keeping assist algorithms. In *Proceedings of the 5th International Conference on Vehicle Technology and Intelligent Transport Systems - Volume 1: VEHITS,*, pages 333–342. INSTICC, SciTePress.

Powell, M. J. D. (2006). The newuoa software for unconstrained optimization without derivatives. *Large Scale Nonlinear Optimization*, page 255–297.

Pütz, A., Zlocki, A., Bock, J., and Eckstein, L. (2017). System validation of highly automated vehicles with a database of relevant traffic scenarios. In *12th ITS European Congress.*

Qi, Y., Li, K., Kong, W., Wang, Y., and Luo, Y. (2019). A trajectory-based method for scenario analysis and test effort reduction for highly automated vehicle. In *WCX SAE World Congress Experience.*

Quinlan, J. R. (1986). Induction of decision trees. *MACH. LEARN*, 1:81–106.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Raina, R., Madhavan, A., and Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*, pages 873–880. ACM.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). Why Should I Trust You?': Explaining the Predictions of Any Classifier. In *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM.

Riedmaier, S., Ponn, T., Ludwig, D., Schick, B., and Diermeyer, F. (2020). Survey on scenario-based safety assessment of automated vehicles. *IEEE Access*, 8:87456–87477.

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

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Route, S. (2017). Test optimization using combinatorial test design: Real-world experience in deployment of combinatorial testing at scale. In *10th IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*.

SAE, I. (2018). Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. Technical report, J3016.

Santiago, J., Claeys-Bruno, M., and Sergent, M. (2012). Construction of space-filling designs using wsp algorithm for high dimensional spaces. In *Chemometrics and Intelligent Laboratory Systems*, volume 113, pages 26–31. Elsevier.

So, J., Park, I., Wee, J., Park, S., and Yun, I. (2019). Generating traffic safety test scenarios for automated vehicles using a big data technique. *KSCE Journal of Civil Engineering*, 23.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Stark, L., Düring, M., Schoenawa, S., Maschke, J. E., and Do, C. M. (2019). Quantifying vision zero: Crash avoidance in rural and motorway accident scenarios by combination of acc, aeb, and lks projected to german accident occurrence. *Traffic Injury Prevention*, 20(sup1):S126–S132. PMID: 31381430.

Stark, L., Obst, S., Schoenawa, S., and Düring, M. (2019). Towards vision zero: Addressing white spots by accident data based adas design and evaluation. In *2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 1–6.

Stellet, J., Zofka, M., Schumacher, J., Schamm, T., Niewels, F., and Zöllner, J. (2015). 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*.

Storn, R. and Price, K. (1997a). Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359.

Storn, R. and Price, K. (1997b). Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359.

Sörensen, K. (2013). Metaheuristics – the metaphor exposed. *International Transactions of Operations Research*, In Press.

Tourbier, Y. (2017). Autonomous vehicle main validation algorithm. Technical report, Renault.

Tuncali, C. E. (2019). *Search-based Test Generation for Automated Driving Systems: From Perception to Control Logic.* PhD thesis, Arizona State University, Tempe.

Tuncali, C. E. and Fainekos, G. (2019). Rapidly-exploring random trees for testing automated vehicles. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 661–666.

Tuncali, C. E., Fainekos, G., Ito, H., and Kapinski, J. (2018). Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1555–1562.

Tuncali, C. E., Fainekos, G., Prokhorov, D. V., Ito, H., and Kapinski, J. (2020). Requirements-driven test generation for autonomous vehicles with machine learning components. *IEEE Transactions on Intelligent Vehicles*, 5:265–280.

Tuncali, C. E., Pavlic, T. P., and Fainekos, G. (2016). Utilizing s-taliro as an automatic test generation framework for autonomous vehicles. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1470–1475.

Tuncali, C. E., Yaghoubi, S., Pavlic, T. P., and Fainekos, G. (2017). Functional gradient descent optimization for automatic test case generation for vehicle controllers. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pages 1059–1064.

Vishnukumar, H. J., Butting, B., Müller, C., and Sax, E. (2017). Machine learning and deep neural network - artificial intelligence core for lab and real-world test and validation for ADAS and autonomous vehicles: AI for efficient and quality test and validation. In *IntelliSys*, pages 714–721.

Wang, J., Zhang, C., Liu, Y., and Zhang, Q. (2018). Traffic sensory data classification by quantifying scenario complexity. In *2018 IEEE Intelligent Vehicles Symposium (IV)*.

Wang, X., Peng, H., and Zhao, D. (2020). Combining reachability analysis and importance sampling for accelerated evaluation of highly automated vehicles at pedestrian crossing. *ASME Letters in Dynamic Systems and Control*, 1:1–6.

Wistuba, M., Rawat, A., and Pedapati, T. (2019). A survey on neural architecture search. *CoRR*, abs/1905.01392.

Xia, Q., Duan, J., Gao, F., Chen, T., and Yang, C. (2017). Automatic generation method of test scenario for adas based on complexity. In *SAE Technical Paper*.

Xia, Q., Duan, J., Gao, F., Hu, Q., and He, Y. (2018). Test scenario design for intelligent driving system ensuring coverage and effectiveness. *International Journal of Automotive Technology*, 19:751–758.

Xie, F., Chen, T., Xia, Q., Huang, L., and Shu, H. (2018). Study on the controlled field test scenarios of automated vehicles. In *SAE Technical Paper*. SAE International.

Zhang, C., Liu, Y., Zhang, Q., and Wang, L. (2018a). A graded offline evaluation framework for intelligent vehicle's cognitive ability. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 320–325.

Zhang, S., Peng, H., Zhao, D., and Tseng, H. E. (2018b). Accelerated evaluation of autonomous vehicles in the lane change scenario based on subset simulation technique. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3935–3940.

Zhao, D. (2016). *Accelerated Evaluation of Automated Vehicles*. PhD thesis, University of Michigan.

Zhao, D., Huang, X., Peng, H., Lam, H., and Leblanc, D. (2016). Accelerated evaluation of automated vehicles in car-following maneuvers. *IEEE Transactions on Intelligent Transportation Systems*, PP.

Zhao, D., Lam, H., Peng, H., Bao, S., LeBlanc, D. J., Nobukawa, K., and Pan, C. S. (2017). Accelerated evaluation of automated vehicles safety in lane-change scenarios based on importance sampling techniques. *IEEE transactions on intelligent transportation systems*, 18(3):595–607.

Zhou, J. and del Re, L. (2017). Reduced complexity safety testing for adas & adf. *IFAC-PapersOnLine*, 50(1):5985 – 5990. 20th IFAC World Congress.

Åsljung, D., Nilsson, J., and Fredriksson, J. (2016). Comparing collision threat measures for verification of autonomous vehicles using extreme value theory. *IFAC-PapersOnLine*, 49(15):57 – 62. 9th IFAC Symposium on Intelligent Autonomous Vehicles IAV 2016.

Åsljung, D., Nilsson, J., and Fredriksson, J. (2017). Using extreme value theory for vehicle level safety validation and implications for autonomous vehicles. *IEEE Transactions on Intelligent Vehicles*, PP:1–1.

Åsljung, D., Westlund, M., and Fredriksson, J. (2019). A probabilistic framework for collision probability estimation and an analysis of the discretization precision. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 52–57.

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)
université PARIS-SACLAY

Titre : Modèles et algorithmes pour l'exploration de l'espace des scénarios: vers la validation du véhicule autonome

Mots clés : Véhicule Autonome, Optimisation, Apprentissage, Réseaux de Neurones, Programmation Mathématique Linéaire, Programmation Génétique

Résumé : Les véhicules autonomes sont des systèmes assez complexes où plusieurs types de défaillances peuvent se produire, enclenchant une fausse action sur la route. Chaque composant devra donc être testé rigoureusement pour anticiper et éliminer les potentielles défaillances. Des techniques de simulation numérique sont utilisées pour complémenter les essais de conduite réelle dans la validation. L'objectif de cette thèse CIFRE est de contribuer des algorithmes et méthodes à un projet industriel chez Renault focalisé sur la validation par simulation de la loi de commande en Model-In-the-Loop (MIL).
Les contributions de cette thèse sont organisées selon trois objectifs: détection d'un nombre maximal de défaillances de la loi de commande, détection de scénarios près de la frontière entre zones défaillantes et sûres, et construction de modèles explicables de la frontière pour l'identifier aussi précisément que possible.
Des techniques d'apprentissage (forêt aléatoire) et d'optimisation (CMA-ES) sont utilisées pour satisfaire la contrainte industrielle de réduire la puissance de calcul, et trois approches sont considérées pour construire des modèles de frontière en analysant leurs performances et explicabilités: réseaux de neurones, programmation mathématique linéaire, et programmation génétique appliquée à la régression symbolique.

Title : Models and algorithms for the exploration of the space of scenarios: toward the validation of the autonomous vehicle

Keywords : Autonomous Vehicle, Optimization, Machine Learning, Deep Learning, Mixed-Integer Linear Programming, Genetic Programming

Abstract : Autonomous vehicles represent highly complex systems, where multiple types of failures could occur leading to a wrong execution on the road. Therefore, each component should be thoroughly tested to anticipate potential failures and mitigate them. Simulation testing methods are used to complement real test driving in the validation process. The aim of this CIFRE PhD thesis is to feed an industrial project at Renault novel algorithms and methods to facilitate the validation of the command law requirements by performing Model-In-the-Loop (MIL) testing.
The main contributions of this PhD thesis are threefold: detecting a maximum number of failures of the command law, detecting scenarios as close as possible to the border separating zones of failed and safe scenarios, and building explainable border models to identify the border as accurately as possible.
The algorithms of the first two objectives use machine learning (Random Forest) and optimization (CMA-ES) techniques to abide with the industrial requirement of reducing the computing power needed, and three approaches are considered to build border models while comparing their performances and explainabilities: Neural Networks, Mixed-Integer Linear Programming (MILP), and Genetic Programming (GP) applied to symbolic regression.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France