COOPERATIVE COLLISION AVOIDANCE FOR AUTONOMOUS VEHICLES USING MONTE CARLO TREE SEARCH

by

Dhruvkumar Patel

APPROVED BY SUPERVISORY COMMITTEE:

Rym Zalila-Wenkstern, Chair

Farokh B. Bastani

John H. L. Hansen

Jessica Ouyang

This dissertation is dedicated to my beloved Guru
Mahant Swami Maharaj, my wife, and my parents
for their endless love, encouragement, and support.

COOPERATIVE COLLISION AVOIDANCE FOR AUTONOMOUS VEHICLES USING MONTE CARLO TREE SEARCH

by

DHRUVKUMAR PATEL, BTech, MS

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

December 2021

ACKNOWLEDGMENTS

I would like to express my profound gratitude to my supervising professor, Dr. Rym Zalila-Wenkstern, who has shown me what I can achieve by pushing me beyond my perceived limits time and again. She has not only been an amazing mentor but also family and a good friend. I cannot express enough thanks to my dissertation committee, Professor Farokh Bastani, Profesh John Hansen, and Professor Jessica Ouyang for their expert and constructive feedback, and for devoting their invaluable time for my proposal and dissertation defense. I am also grateful to previous graduate students from UTD MAVS lab, especially Dr. Behnam Torabi and Dr. Mohammad Al-Zinati, who helped me with their expertise whenever I needed. Finally, I want to express my gratitude to my friends who made the challenging journey a pleasent experience for me.

December 2021

COOPERATIVE COLLISION AVOIDANCE FOR AUTONOMOUS VEHICLES USING MONTE CARLO TREE SEARCH

Dhruvkumar Patel, PhD
The University of Texas at Dallas, 2021

Supervising Professor: Rym Zalila-Wenkstern, Chair

Autonomous vehicles require an effective cooperative action planning strategy in an emergency situation. Most action planning approaches for autonomous vehicles do not scale well with the number of vehicles. In this dissertation, we present COCOA (Cooperative Collision Avoidance), an efficient cooperative action planning algorithm for autonomous vehicles in colliding situations. In COCOA, autonomous vehicles drive together in coalition formations for information sharing and cooperation. When a coalition member detects a colliding situation with a misbehaving vehicle, all coalition members explicitly cooperate to find conflict-free action plans to avoid collisions with the misbehaving vehicle. COCOA employs a hierarchical decision-making approach where action planning is achieved at two levels: at the vehicle level and at the coalition level. In emergency scenarios involving multiple coalitions, COCOA employs a sequential and hierarchical decision-making approach. Leaders of the coalitions in a coalition sequence cooperate to finalize action plans for their coalition members that are free of inter-coalition conflicts. The COCOA algorithm is validated through extensive realistic simulations in a multi-agent-based traffic simulation system.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Ever since the 2004 DARPA Grand Challenge, much effort has been spent developing advanced sensor technologies and smart control systems to achieve high driving automation levels. In 2014, the Society of Automotive Engineers established six driving automation levels documented in the J3016 standard (Committee, 2016). At Level *0*, a vehicle is human-driven and has no autonomous capabilities. At Level *4*, a vehicle is fully autonomous but can operate only in certain conditions (e.g., set areas, good weather conditions). At Level *5*, a vehicle is fully autonomous and can drive everywhere and in all weather conditions.

While existing autonomous vehicles (e.g., Tesla, Mercedes, GM, BMW, Audi) are Levels *2* and *3*, researchers, manufacturers, lawmakers, and regulatory agencies are diligently working on addressing the massive technological, regulatory, and social hurdles to achieve Levels *4* and *5* (Milakis, 2019; Johnson, 2017).

This work focuses on the study of Level 4 Connected and Autonomous Vehicles (CAVs). A CAV is an autonomous vehicle capable of interacting and communicating with other CAVs through short-range communication. CAVs dynamically form coalitions when they are in proximity of each other. Coalition formation supports information sharing among the coalition members to improve roadway safety (Siegel et al., 2018). We focus on CAVs driving on highways and consider the case where a random CAV misbehaves due to technical problems or unexpected environmental changes (e.g., roadblocks, torrential rains, etc.). This research aims to define coalition-based cooperative planning strategies to avoid collisions.

The problem of cooperative action planning for CAVs collision avoidance has been studied in the literature. (Schwarting et al., 2018) extensively reviews many recent approaches for cooperative action planning and motion planning for autonomous vehicles. AI-based and non-AI-based methods have been proposed. AI-based approaches (Lenz et al., 2016),(Kurzer et al., 018a),(Kurzer et al., 018b) generally formulate the planning process as a Markov

Decision Process (MDP). MDPs require enumeration over states and actions making them impractical for large state spaces such as the CAV planning problem. The Monte Carlo Tree Search (MCTS) (Coulom, 2006; Vodopivec et al., 2017) has recently shown promising results in problems with very large state spaces (Silver et al., 2017), but is generally limited in the number of agents.

We present **Co**operative **Co**llision **A**voidance (COCOA), an MCTS-based algorithm for CAV cooperative planning in colliding situations. In COCOA, a plan is defined as a *sequence of actions* for a *planning horizon* and takes into account the continuously exchanged information between coalition members. In this dissertation, we discuss COCOA in two contexts: a single coalition and multiple coalitions. For single coalition decision-making, we propose a two-step decision-making approach. When a misbehaving vehicle is detected, each CAV uses the information exchanged with the coalition CAVs and executes COCOA to derive a set of action plans to avoid collisions. Each CAV's prioritized set of plans is sent to the coalition leader, and in the second step, the leader selects the individual coordinated action plans that are conflict-free, when possible.

For multiple coalitions decision-making, we propose a sequential decision-making approach. When a misbehaving vehicle is detected, the affected coalition executes COCOA algorithm for a single coalition described above to generate action plans for its members and sends these plans to its neighboring coalition. The neighboring coalition takes into account the received plans of the previous coalition when generating plans for its members. These sequential decision-making steps are carried out until the last coalition on the road has finished generating plans for its members. The proposed algorithm is the first cooperative collision avoidance algorithm that is applied to multiple coalitions. It improves upon the state-of-the-art MCTS-based algorithms for CAVs by exponentially reducing the size of the search tree and therefore allows for a scalable solution.

In the next chapter, we give the background knowledge for the topic of our research and discuss related works. In Chapter 3, we formalize the problem and present the COCOA

algorithms for a single coalition. In Chapter 4, we present a case study to examine the behavior of COCOA algorithms for a single coalition. In Chapter 5, we evaluate the algorithms for a single coalition using simulation experiments and present the results. In Chapter 6, we present the COCOA algorithms for multiple coalitions, and in Chapter 7, we evaluate the algorithms for multiple coalitions using simulation experiments.

CHAPTER 2

BACKGROUND

In this chapter, we give background knowledge on CAVs (Patel and Zalila-Wenkstern, 020b) and discuss related works.

2.1 Autonomous Vehicle Technologies

A CAV is a complex automated system that integrates computing, communication, and control technologies (Elliott et al., 2019). At a high level, a CAV must

1. be aware of its surroundings. Therefore it should be able to a) sense the environment in which it is situated, and b) localize itself;
2. be able to communicate with adjacent vehicles and vehicles within the coalition;
3. be able to communicate with the infrastructure to obtain information such as road conditions or emergency situations;
4. continuously assess risks and make autonomous decisions.

In order to fulfill these requirements, a CAV makes use of advanced sensing devices and communication technologies (see Figure 2.1).

2.1.1 Technologies for Environment Sensing

The commonly used technologies for environment sensing include Ultrasonic sensors, RADAR, LiDAR, and cameras. *Ultrasonic sensors* are used as vehicular sensors for near object detection. They provide direct distance measurements for very near range (i.e., up to 2 meters) (Rasshofer and Gresser, 2005). The main advantage of ultrasonic sensors is their ability to operate in poor weather conditions. *LiDAR* and *RADAR* are long-range detection systems used to determine the distance, angle, and speed of vehicles, pedestrians, and obstacles (Zhao

Figure 2.1: CAV Sensing and Communication Technologies

et al., 2018). A LiDAR system can detect objects with higher accuracy than a RADAR but its detection range is shorter and costs more (i.e., tens of meters for LiDAR versus up to 200 meters for RADAR) (Rasshofer and Gresser, 2005). Google, Uber, and Toyota rely heavily on LiDARs. Tesla uses RADARs with cameras (Rasshofer and Gresser, 2005). Latest developments in LiDAR technologies have brought the costs down and have increased its range significantly making it comparable to the RADAR range (Hawkins, 2020).

2.1.2 Technologies for Localization

A localization system identifies the location of a CAV on a global coordinate system. While often inaccurate and unreliable, GPS is currently the most commonly used localization system for CAVs, mainly due to its low cost and high availability. To address GPS limitations companies have used additional technologies such as Inertial Motion Units (IMU) (Zhang et al., 2012). The error of GPS integrated with IMU was found to be the root mean square

value of 7.2m, compared to 13.2m for GPS (Zhang et al., 2012). However, GPS integrated with IMU is prone to error accumulation as the vehicle travels along its desired trajectory. Therefore, the use of devices such as LIDAR may lead to more reliable results (Kuutti et al., 2018). Researchers have also suggested the use of communication data to improve localization accuracy (Kuutti et al., 2018).

2.1.3 Technologies for Vehicle Communications

Vehicular communication is a core requirement for CAVs. In order to provide Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) capabilities, three main types of networking approaches are commonly used (Siegel et al., 2018). These include *mesh networking*, *cellular networking* and *hybrid networking*. We discuss each of the approaches below.

Mesh networking

A mesh network is a distributed network topology in which nodes connect with each other directly and dynamically to efficiently route data to/from clients (Djukic and Valaee, 2018). A vehicular ad-hoc network (VANET) is a vehicular mesh network that allows moving vehicles to rapidly exchange safety messages with one another and with the traffic infrastructure (Bhatia et al., 2018).

The key enabling communication technology for VANET is Dedicated Short Range Communication (DSRC). DSRC is an open-source protocol for secure, high-speed wireless communications. It supports high-speed data rates (up to 27 MB/s) and has a transmission range of 100 to 1000 meters. Additionally, its low latency makes it a preferred wireless technology for road safety messaging. In the US, the Federal Communications Commission has allocated 75 MHz of spectrum in the 5.9 GHz frequency to be used for mobility applications. DSRC operates in this band and has been standardized as a communication protocol for a variety of applications in Intelligent Transportation Systems (ITS). Nevertheless, DSRC

is limited by intermittent connectivity due to fast-moving vehicles and changing VANET topologies (Abboud et al., 2016).

Cellular Networking

The latest development in cellular vehicular networking is Cellular Vehicle-to-Everything (C-V2X) technology. C-V2X is being developed as part of the overall 3^{rd} Generation Partnership Project (Naik et al., 2019) to advance cellular systems from 4G to 5G technologies. 3GPP is a union of seven telecommunications organizations, which work on next-generation 5G standards. C-V2X developers plan to enable every DSRC capability in C-V2X and expand them even further. C-V2X has a clear advantage over DSRC in V2I communications, since it can leverage traditional cellular infrastructures and, unlike DSRC, does not rely on installing new devices on the road networks for V2I capabilities (Ghafoor et al., 2019). According to testing done by Qualcomm, C-V2X doubles the range of DSRC and increases the alert time by a few seconds (Lucero, 2016).

Hybrid Networking

Several hybrid vehicular networking architectures have been proposed. The guidelines of Communication Access for Land Mobile (Ernst et al., 2009) discusses a vehicular communication infrastructure that combines VANETs and centralized cellular networks. The hybrid solution leads to a broader coverage area and high networking throughput. Other hybrid solutions include the integration of mobile cloud computing and context-aware technologies (Ahmad et al., 2017). These solutions have not been widely implemented and tested.

In this research, we consider a CAV equipped with the following:

- Short-range LiDAR that covers about 100 meters in all directions in a 360-degree field-of-view.

- Long-range RADARs in front and back which cover the equivalent of 200 meters in an 18-degree field-of-view. The range and field-of-view values can be adjusted dynamically to simulate medium-range.
- GPS for localization.
- DSRC for continuous communication with other CAVs within a maximum of 500-meter range. This range can be dynamically changed. The continuously shared information with surrounding vehicles includes vehicle position, heading, speed, and dimensions.

This configuration is the technical basis for the state-of-the-art CAVs available to date (e.g., Mercedes S500 and Audi A8).

2.2 Autonomous Vehicle Simulation

An agent-based simulation environment that supports sensing, autonomy, communication, and decentralization is critical for a realistic assessment of autonomous vehicle algorithms. We evaluate our algorithms in MATISSE 3.0, a microscopic multi-agent-based traffic simulation system developed at the MAVS lab at UT Dallas (Torabi et al., 2018). In MATISSE, vehicles and intersection controllers are modeled as virtual agents which perceive their surroundings through sensors and continuously interact with one another.

The virtual CAV is implemented as an autonomous decision-making agent. It can dynamically sense the surrounding environment and other CAVs using simulated short and long-range sensors (see Figure 2.3). It can also communicate with other virtual CAVs within its communication radius through simulated V2V communication (see Figure 2.2). During the simulation, the user can modify a CAV's properties (e.g., sensor range, communication radius, etc.) and behavior (e.g., speed) and witness the outcome in simulated real-time.

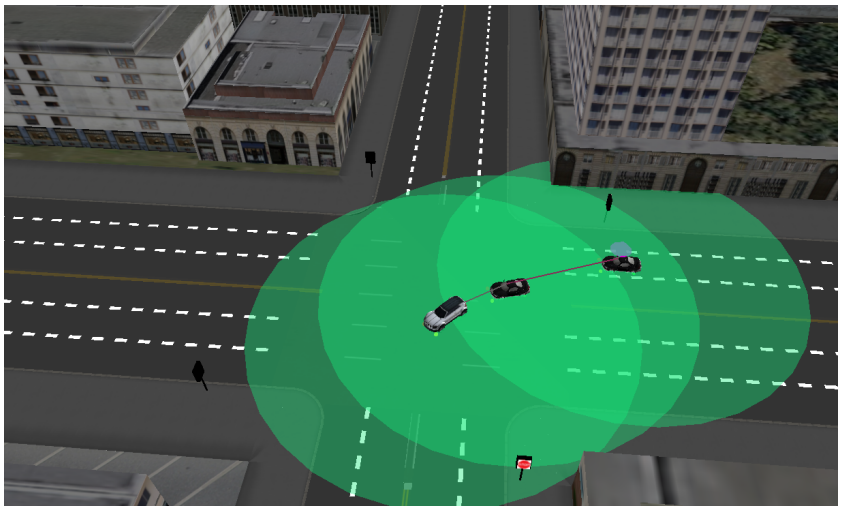


Figure 2.2: Vehicle agents communicate within communication radius in MATISSE

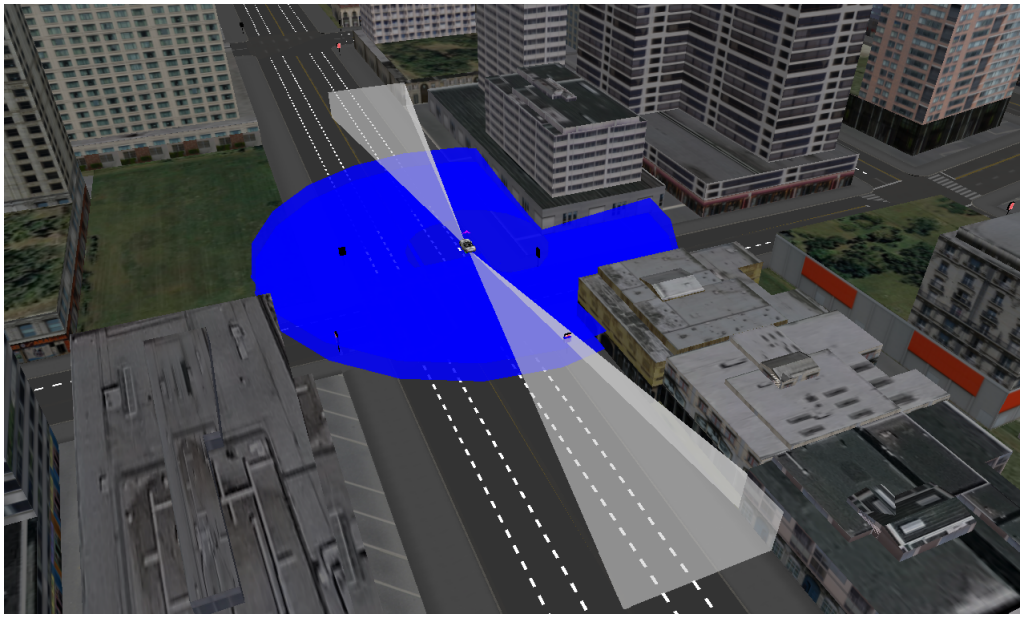


Figure 2.3: Vehicle agent with long range RADAR and short range LiDAR in MATISSE

2.3 Autonomous Vehicle Action Planning

Autonomous vehicle action planning is the task of determining the next CAV actions (e.g., speed up, brake, lane change etc.) in any given CAV state. A CAV state is a vector containing CAV's position, speed, steering angle, dimensions etc. Since the state space for

CAVs is extremely large, classical planning algorithms are not very efficient for the CAV action planning problem (Andriotis and Papakonstantinou, 2019). Reinforcement learning techniques from the AI domain have traditionally been used to solve different multi-agent action planning problems. We explore a reinforcement learning technique known as Monte Carlo Tree Search (MCTS) (Vodopivec et al., 2017), that has recently shown promising results in the problems with very large state spaces such as the game of Go (Silver et al., 2017).

MCTS is a simulation-based tree-search algorithm. Each CAV executes MCTS to build a search tree iteratively (Lan et al., 2020). Nodes in the MCTS tree correspond to the CAV states and edges correspond to the CAV actions. Each tree node stores two numerical parameters, known as *node value* and *node count*. A CAV executes MCTS by first creating a single node tree that stores the current CAV state and then performs many MCTS iterations to explore this tree further. A single MCTS iteration consists of four steps (See Figure 2.4): *Selection*, *Expansion*, *Simulation*, and *Backpropagation*. MCTS makes use of a reward function during its simulations. CAV actions that lead to colliding CAV states receive a negative/lower reward, whereas CAV actions that lead to desirable CAV states receive a positive/higher reward. Below we describe each of the above-mentioned four steps of a single MCTS iteration:

- Selection: In the selection step, CAV starts from the root node and selects consequent child nodes using some heuristic function defined over the node parameters (node value and node count) until a leaf node is reached.

- Expansion: In the expansion step, CAV expands the leaf node by adding all possible child nodes at that leaf node. Each child node corresponds to a distinct CAV action available at the leaf node.

- Simulation: In the simulation step, CAV selects one of the expanded child nodes and performs a Monte-Carlo simulation of CAV actions starting from that child node.

- Backpropagation: In the backpropagation step, the reward received at the end of the simulation step is used to update the node values and the node counts in a reverse-path from the leaf node to the root node.

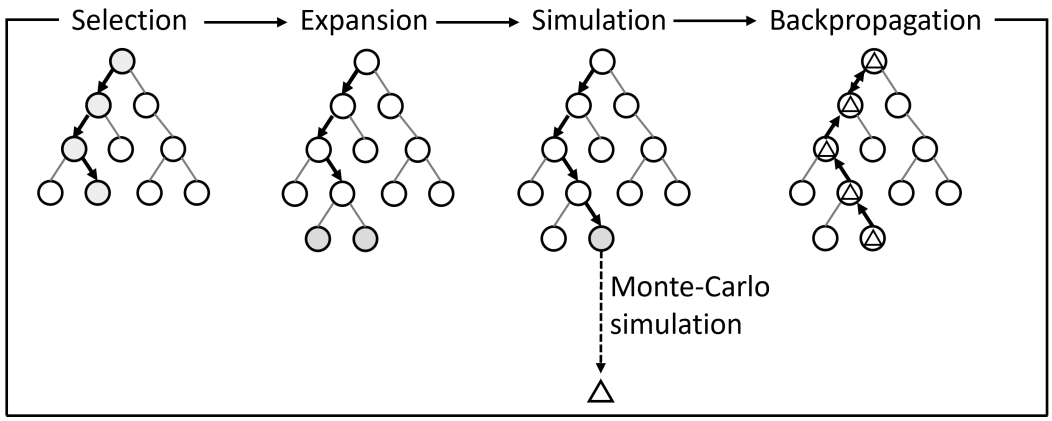


Figure 2.4: Four steps in a single iteration of Monte Carlo Tree Search

The resultant MCTS tree can be used to select CAV's next best action. Although the basic version of MCTS handles the problem of a large CAV state-space, it does not scale well as the number of CAVs grows. Additionally, it does not guarantee the collision-free solution of CAV actions in some cases even if one exists (Schaeffer et al., 2009). As such, it needs to be significantly improved upon for its application to the CAV action planning problem.

2.4 Related works

Many CAV collision avoidance methods have been proposed in the literature. We classify these methods according to the decision-making (i.e., planning) approach, namely *non AI-based optimization* and *AI-based optimization* and further categorize them as *centralized* or *decentralized*. Centralized CAV collision avoidance systems are systems in which a centralized computer is responsible for making collision avoidance decisions for vehicles. In decentralized systems, the decision-making is performed individually by each CAV.

2.4.1 Non AI-based approaches

Centralized

Most conventional optimization-based approaches fall into this category. These approaches formulate the trajectory planning problem for CAVs as a single, global optimization problem. We list these approaches in Table 2.1. In these approaches, each vehicle typically has partial knowledge of the environment. Each vehicle shares its state and/or trajectory information with the centralized server. The centralized server has full knowledge of the environment. The server uses the proposed optimization algorithm to find non-colliding trajectories for all vehicles. The main difference is in the optimization method used by the server.

(Kessler and Knoll, 2019) proposes to use Mixed Integer Linear Programming (MILP) with the collision constraints added to the MILP problem. In (Burger and Lauer, 2018), authors propose to use Mixed Integer Quadratic Programming (MIQP) with the collision constraints added to the MIQP problem. The main limitation of these approaches is their inability to provide a solution in practically feasible computation time, especially in scenarios with a large number of vehicles. (Schwarting and Pascheka, 2014) proposes to use a two-step method to generate collision-free actions. In the first step, an egoistic action for each CAV is derived. For each pair of CAVs that has conflicts, the lowest cost action combination is chosen in the second step by generating all maneuver combinations. (Duering et al., 2014) proposes a graph theory-based optimization approach. Each vehicle defines its reachable target points (RTPs) and sends them to the server. The server checks these points for collisions with other vehicles and static objects and defines safe target points (STPs), from which final trajectories are generated using grid generation, path planning, and trajectory planning. The main limitation of this approach is the large computation time required due to high RTP density, a high number of STPs, and a high number of trajectory combinations. (Manzinger et al., 2017) proposes to use a pre-defined maneuver template approach. A

maneuver template describes the continuous dynamics of a set of vehicles along with several constraints. The server checks the off-line calculated maneuver templates to match with the current traffic scene with specific initial and feasibility constraints. The maneuver template with the lowest cost specifies the cooperative maneuver. The main limitation of this approach is that it may require a very large number of off-line calculated maneuver templates to cover all possible traffic situations. In (Wang et al., 2018), the authors propose a cloud computing-based solution that uses parallel processing. The vehicle trajectory planning problem is formulated as a centralized Quadratic Programming (QP) problem. The authors apply the Alternating Direction Method of Multipliers (ADMM) (Boyd et al., 2011) to decompose the centralized optimization problem. A network of computing nodes solves the decomposed centralized optimization problem. The results show that the parallel processing method is slower than the same method without parallelization for up to 25 vehicles.

Nakamura et al. (2020) presents a short-term trajectory generation algorithm with the formal guarantee of collision avoidance using an SAT (Satisfiability) solver. A central server formulates the collision avoidance problem as a constrained integer programming problem. In this formulation, trajectories are discretized and represented using a set of waypoints. Each waypoint represents the vehicle status, i.e., position, velocity, and acceleration (laterally and longitudinally) at a discrete time step. Optimization variables are boolean variables that represent if a vehicle should accelerate or not at each time step. These boolean variables are defined for both lateral and longitudinal directions. Constraints for collision avoidance between two vehicles, constraints for collision avoidance between a vehicle and an obstacle, and constraints to reach a specific goal state are added to the integer programming problem. The optimization objective is defined as the minimization of the travel time. This optimization problem is solved using CP-SAT solver from Google OR-Tools software. Two methods are introduced to reduce the computation time. In the first method known as the grouping method, vehicles are grouped into disjoint sets of interacting vehicles. Trajectory

Table 2.1: Non-AI based Centralized Approaches

Approach	Knowledge				Data acquisition				Optimization method
	Central		CAVs		Central	CAVs			
	Full	Part.	Full	Part.	V2I	V2V	V2I	Sensors	
(Kessler and Knoll, 2019)	✓			✓	✓	✓		✓	MILP
(Duering et al., 2014)	✓			✓		✓			Graph theory
(Schwarting and Pascheka, 2014)	✓			✓					Exhaustive search
(Burger and Lauer, 2018)	✓			✓	✓	✓			MIQP
(Manzinger et al., 2017)	✓			✓					Maneuver template
(Wang et al., 2018)	✓			✓	✓			✓	ADMM
(Nakamura et al., 2020)	✓			✓	✓	✓			SAT

optimization is performed for each disjoint set separately. In the second method known as the collision checkpoint method, optimization is carried out after every specific number of discrete time steps instead of after every time step. Knowledge of the dynamic environment including all vehicles states is acquired using V2V and V2I communication. The main limitation of this approach is its inability to provide a solution in real-time, especially in scenarios with a large number of vehicles.

We consider (Nakamura et al., 2020) state-of-the-art in centralized approaches, as it gives the formal guarantee of collision avoidance whenever possible. Additionally, it also reduces computation time by 98% using the collision checkpoint method compared to using

Table 2.2: Non-AI based Decentralized Approaches

Approach	CAV knowledge		CAV data acquisition				Optimization method
	Full	Partial	V2V	V2I	Sensors	Not mentioned	
(Duering and Pascheka, 2014)	✓		✓				Exhaustive search
(Krajewski et al., 2016)		✓	✓	✓			Graph theory
(Gao et al., 2019)	✓		✓		✓		Potential field
(Bellan and Wartnaby, 2020)		✓	✓		✓		Nonlinear optimization

the centralized approach without this method. Although centralized approaches provide optimal solutions, they are computationally expensive and not scalable.

Decentralized

(Duering and Pascheka, 2014) presents a brute force algorithm that performs an exhaustive search on all maneuver combinations for all CAVs. Each CAV derives all possible maneuvers for itself and associates cost for each maneuver using a cost function. The list of maneuvers and associated costs are shared with all other CAVs using V2V communication. Each CAV then goes over all possible joint combinations of maneuvers to find the collision-free combination with the lowest cost. Decision-making is performed at the vehicle level. Each CAV has full knowledge of the static and dynamic environment. Knowledge of the dynamic environment is acquired using V2V communication. Coordination is achieved explicitly using V2V communication. The main limitation of this approach is its inability to handle a large number of CAVS and the fact that the solution is not better than any centralized approach as the same exhaustive computation is carried out by each CAV. In their later work

(Pascheka and Duering, 2015), authors added the "memory of cost" parameter for each CAV without changing the core algorithm. This parameter helps with the cases when one CAV is consecutively preferred over the other in multiple decision-making steps. (Pascheka and Duering, 2015) has same limitations as (Duering and Pascheka, 2014).

(Krajewski et al., 2016) proposes a graph theory-based approach. A two-step method based on the A* graph search algorithm is proposed for collision avoidance at urban intersections. In the Trajectory Layer (TL) step, each CAV computes its unique trajectory using A* search to minimize the cost. In the Negotiation Layer (NL) step, all CAVs approaching the intersection share their trajectories with each other. Each CAV decides its own group based on the conflicting trajectories. Each CAV performs A* search again to compute a new trajectory with the collision avoidance constraints. This two-step process is repeated until the collision-free trajectories are found. The decision-making is performed at the vehicle level in this approach. Each CAV has access to full knowledge about the static environment and partial knowledge about the dynamic environment. Knowledge about other vehicles and the static environment is acquired using V2V and V2I communication. Coordination is achieved explicitly between CAVs using V2V communication. The main limitation of this approach is that vehicle groups are created at the beginning of the algorithm based on collisions detected at the beginning but they are not updated later. In subsequent phases, new collisions may occur between CAVs of different groups due to the updated trajectories.

(Gao et al., 2019) proposes an artificial potential field approach for a convoy of CAVs. Each CAV formulates the environment representation that contains an artificial repulsive potential function for each of the CAVs and obstacles. Each CAV uses a gradient descent method to derive its trajectory in this environment of repulsive potential functions that allow the CAV to avoid collisions with the other CAVs and the obstacles. However, this approach does not induce a complex cooperative behavior needed for many conflicting scenarios.

(Bellan and Wartnaby, 2020) proposes a desired-versus-planned approach. Each CAV derives and broadcasts two trajectories: a planned trajectory that it is currently following

Table 2.3: AI based (Decentralized) Approaches

Approach	CAV knowledge		CAV data acquisition				Optimization method
	Full	Partial	V2V	V2I	Sensors	Not mentioned	
(Lenz et al., 2016)		✓			✓		MCTS
(Kurzer et al., 018a)		✓			✓		MCTS
(Kurzer et al., 018b)		✓			✓		MCTS
(Kurzer et al., 2020)		✓			✓		MCTS
(Kurzer et al., 018b)		✓	✓		✓		Actor-critic
COCOA		✓	✓		✓		MCTS

and the desired trajectory that it wants to follow. When a different CAV receives this information, it checks if it can accommodate the first CAV's desired trajectory and broadcasts its updated trajectory. A cost-based non-linear optimization problem is solved to generate the updated trajectories. The authors tested this approach for two CAVs in an actual road test. Although this approach is successful for two CAVs in simple conflicting situations, it is not scalable to more than two CAVs as the algorithm takes many replanning steps for each CAV to converge.

2.4.2 AI based approaches

In AI-based approaches, each CAV performs decision-making by learning the value of an action in its current state. Systems in this category are decentralized, i.e., each CAV determines the value of available actions individually using implicit or explicit cooperation with other CAVs.

(Lenz et al., 2016) proposes a decision-making approach based on Monte-Carlo Tree Search (MCTS). A CAV has access to full knowledge of the road network and acquires data about other CAVs using sensors. Each CAV performs an MCTS algorithm which includes the classical four steps, i.e., *selection*, *expansion*, *simulation* and *backpropagation* (Browne et al., 2012). In the search tree, a node refers to a joint state of all CAVs, an edge refers to a single CAV action, and a sequence of edges from the root node corresponds to actions taken by all CAVs. At the end of the algorithm, each CAV executes the action having the highest action value at the root node. The approach was evaluated on scenarios involving up to three cooperative vehicles. The main limitation of this approach is scalability. For a larger number of CAVs, the MCTS tree grows exponentially in depth and becomes computationally expensive.

(Kurzer et al., 018a) applies MCTS to cooperative driving formulated as a Semi Markov Decision Process (SMDP). Similarly to (Lenz et al., 2016), each CAV executes the classical four steps of MCTS. A node in the tree refers to a joint state of all CAVs and an edge refers to joint action. To reduce the tree depth, this work uses *macro actions* defined as sequences of primitive actions. For each macro action, a policy determines the start state and the number of primitive actions to be executed before it ends. In the MCTS *selection*, *expansion* and *simulation* steps, all simulated CAVs select macro actions according to their policies until primitive actions are chosen, and then determine the next joint state using joint primitive actions. In the *backpropagation* stage, the value of each node of the tree is updated using the cumulative reward. Each CAV chooses a macro action according to the maximum reward at the root of the tree. The approach was evaluated on scenarios involving up to three CAVs. Similarly to (Lenz et al., 2016), the main limitation of this approach is scalability as the MCTS tree grows exponentially, but in width, for a larger number of CAVs.

(Kurzer et al., 018b) uses a version of MCTS algorithm known as decoupled-UCT (Tak et al., 2014) to solve cooperative driving problem formulated as a decentralized Markov Decision Process (Dec-MDP). Vehicle trajectories are approximated using quintic polynomials and the continuous action-space is not discretized. Each CAV executes the classical four steps of MCTS. A few extensions are introduced to the MCTS algorithm to deal with problems that arise from the large size of the continuous action space. (1) Semantic grouping method allows each vehicle to group its actions at each state into nine action groups. During the MCTS selection step, an action group is selected first at each state and then a vehicle action is selected from the selected action group. (2) In the progressive widening method, each vehicle starts with a small size of predefined action-space at each state initially. As the state is visited more often during the execution of MCTS, more actions are added to the state's action-space using a guided search method. (3) Authors introduce the kernel update method to mitigate the problem of not visiting all states in the tree due to the large action space. This method is used during the backpropagation phase. In this method, the node values and the visit counts of all the states similar to the current state are also updated after being weighted by a state similarity measure. This approach was evaluated on two road scenarios involving up to three cooperative vehicles. The main limitation of this approach is its scalability as the action space of each state and nodes in the MCTS tree grow significantly with the number of vehicles.

In their later work (Kurzer et al., 2020), authors propose a preprocessing step for MCTS. A heuristic model over vehicle actions is learned from a synthetic driving dataset. The learned model is plugged back into DeCoC-MCTS to steer the algorithm towards more promising areas of the action space. The authors show that MCTS with the heuristic model has a higher success rate than the baseline MCTS in some scenarios.

(Yuan et al., 2020) proposes an actor-critic-based method called Cooperative Deep Deterministic Policy Gradients (Co-DDPG) to solve the MMDP. The authors propose that

CAVs share the learned MMDP parameters with each other to improve the collision avoidance success rate. The algorithm was tested on scenarios involving up to three CAVs. The main drawback with the decentralized reinforcement learning-based approaches that have an individual reward function with implicit cooperation is that the collision avoidance is only guaranteed when all CAVs reach Nash equilibrium, i.e., they all find the same solution. Another drawback is scalability in the number of agents.

We consider (Kurzer et al., 018b) state-of-the-art in AI-based decentralized approaches as it uses continuous action-spaces for vehicles as opposed to a pre-defined finite set of actions. They also introduce several extensions to MCTS to overcome the problems arising due to the large size of continuous action spaces. This allows for higher flexibility for vehicle trajectories using continuous actions without increasing the computational complexity of MCTS. The main limitation of current MCTS-based approaches is their limited scalability in the number of CAVs. Another major limitation of MCTS is that Nash equilibrium is necessary for a collision-free solution, but is not always guaranteed. This leads to collisions in many situations.

In this dissertation, we present **Co**operative **Co**llision **A**voidance (COCOA) algorithm for the cooperative action planning for coalitions of CAVs in colliding situations. We use a partially-decentralized approach for the COCOA algorithm. We propose a hierarchical decision-making method where decision-making is performed at two levels: at the CAV level and at the coalition level. Cooperation is achieved explicitly among CAVs in a coalition using V2V communication. Additionally, cooperation among different coalitions is also achieved explicitly using V2V communication. The unique contributions of our approach are:

- It is the first algorithm in our knowledge to achieve cooperative collision avoidance for multiple coalitions of CAVs.

- It introduces a novel hierarchical decision-making method for the cooperative action planning of CAVs in a single coalition with explicit cooperation through V2V communication.

- It introduces a novel sequential decision-making method for the cooperative action planning of multiple coalitions with explicit cooperation through V2V communication.

- It introduces a branching factor reduction method for MCTS that reduces MCTS tree size exponentially to improve scalability (Patel and Zalila-Wenkstern, 020a).

- It introduces an intelligent action selection strategy for MCTS to improve reliability (Patel and Zalila-Wenkstern, 021a).

- It introduces a novel idea of updating the weights of the CAVs reward functions in MCTS dynamically based on coalition states (Patel and Zalila-Wenkstern, 021b) to further improve reliability.

- It uses a simple coalition-level decision making algorithm to find a non-colliding sequence of actions for a planning horizon when possible as opposed to one action at a time, thus removing the possibility of inevitable future collisions.

- It has been thoroughly tested and evaluated against state-of-the-art approaches in MATISSE 3.0, a large-scale multi-agent-based simulation system.

CHAPTER 3

COCOA ALGORITHMS FOR A SINGLE COALITION

3.1 Model Definition

In the remainder of this dissertation, the terms *agent* and *CAV* are used interchangeably. We assume that a coalition $\mathcal{C}$ of n CAVs is navigating on a straight highway. The coalition is formed when a finite set of CAVs are in close proximity. $\mathcal{C}$'s agents are called *coalition members*. One coalition member is assigned the role of a *leader* and is responsible for coalition management. Coalition formation and leader assignment algorithms are adapted from (Manoochehri and Wenkstern, 2017).

A CAV is defined by its ID $i \in \mathcal{C}$ and a state vector $s^i = [p^i, v^i, l^i, \theta^i, x^i]$ where p^i is the CAV's position coordinates, v^i is its velocity, l^i is its current lane, θ^i is its orientation, and x^i is the size of the vehicle containing length and width values. A coalition *joint state* $s = \{s^i\}_{i \in \mathcal{C}}$ is the set of states of all coalition members.

Each CAV can perform the actions listed in Table 3.1. The effect of taking an action a^i_t at time t is determined using the trajectory of i, denoted as $\tau^i_{a^i_t}$. $\tau^i_{a^i_t}$ is computed using velocity, position and orientation. A *mitigation action plan* α^i_h is a sequence of h consecutive actions that is defined by a CAV i in case of perceived danger or warning related to a misbehaving vehicle. It is defined as $\alpha^i_h = \{a^i_{t_k}\}^h_{k=1}$, where t_k is the start of the execution of the mitigation plan's k'th action. h is known as the *planning horizon*. An action is performed over a duration denoted as Δt.

As mentioned in Chapter 2, CAVs in a coalition use V2V communication to continuously exchange information with each other. The information exchanged depends on the CAV role in the coalition (i.e., member or leader) and includes a CAV's state s^i, the coalition joint state $s = \{s^i\}_{i \in \mathcal{C}}$, a warning or an action plan α^i_h.

Table 3.1: CAV Actions

Action	Condition	Description
maintain	-	Maintain the current velocity
accel	-	Accelerate with a fixed acceleration value α_{acc}
decel	-	Decelerate with a fixed deceleration value α_{dec}
cll	Currently not in the left most lane	Change lane to the left lane of the current lane
clr	Currently not in the right most lane	Change lane to the right lane of the current lane

3.2 Hierarchical Approach

In an event where a misbehaving vehicle is detected, the coalition needs to derive conflict-free mitigation action plans for all coalition members to avoid collision with the misbehaving vehicle. To that end, we propose a hierarchical and partially-decentralized decision-making approach where decision-making is performed at two levels: at the CAV level and the coalition level. In the first step, each CAV i independently performs decision-making to find a set of prioritized individual mitigation action plans $\{\alpha_h^i\}$ and sends these plans to the coalition leader. In the second step, the coalition leader analyzes each CAV's prioritized mitigation action plans and deliberates to find, an action plan α_h^i for each CAV i such that the collision constraints are satisfied when possible.

3.2.1 CAV level approach

The first step of our approach involves decision-making at the individual CAV level to find prioritized individual action plans. Each CAV i independently performs decision-making to find a set of prioritized individual mitigation action plans $\{\alpha_h^i\}$. We formulate the problem of decentralized individual action planning for each CAV as a Multi-agent Markov Decision

Process (MMDP) (Boutilier, 1999). Unlike a conventional MMDP where each agent considers the immediate reward for joint actions, our approach focuses on maximizing the overall coalition reward based on CAVs' individual action plans for a horizon. MMDP is defined as a tuple $\langle \mathcal{C}, S, A, T, R, \gamma \rangle$, where

- $\mathcal{C}$ is a coalition of n CAVs.
- S represents the joint state space for the CAVs in $\mathcal{C}$. $S = \times S^i$ where S^i is the state space for CAV i.
- A represents the joint action space for the actions of CAVs in $\mathcal{C}$. $A = \times A^i$ where A^i is the set of actions that i can perform.
- $T : S \times A \times S \rightarrow [0, 1]$ is the transition function where $T(s, a, s')$ specifies the probability of the system transitioning to state s' when performing joint action a in state s.
- $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function with $R(s, a, s')$ specifying the reward received when executing joint action a in state s and transitioning to the state s'.
- γ is a discount factor controlling the contribution of future rewards on the current state.

Generally, the solution to an MMDP are state-action values computed by function $Q^* : S \times A \rightarrow \mathbb{R}$. $Q^*(s, a)$ value represents the expected reward of taking the joint action a in the coalition joint state s. In any given state s, an optimal action is an action that has the maximum $Q^*(s, a)$ value. Each CAV executes Monte Carlo Tree Search (MCTS) based algorithms (see Section 3.3) to solve the MMDP and to estimate the Q^* values in order to prioritize its mitigation action plans $\{\alpha_h^i\}$.

MCTS for CAV level action planning

Monte Carlo Tree Search (MCTS) is a simulation-based search algorithm to solve MMDP with the help of random samples (see Section 2.3). Each CAV uses MCTS to solve the MMDP in order to prioritize its individual mitigation action plans. Each CAV $i \in \mathcal{C}$ individually executes the MCTS algorithm to build a search tree iteratively. In the tree, each node represents a coalition joint state s and each edge represents a coalition joint action a. The average number of children at each tree node is known as *branching factor*. For our problem, the branching factor is the size of the joint action space, which equals $|A^i|^n$. The objective for each CAV is to find m non-colliding mitigation action plans of size h. To select h consecutive actions, each CAV executes the MCTS algorithm to construct an MCTS tree of height h. Each path in the tree from the root node up to the leaf node represents a possible mitigation action plan. MCTS tree should be sufficiently explored up to depth h to select m non-colliding action plans, as there should be at least m distinct paths from the root node to leaf nodes. The tree with the branching factor $|A^i|^n$ and the tree depth h has a total of $\frac{|A^i|^{n(h+1)}-1}{|A^i|^n-1}$ nodes. Here, we can see that the tree size exponentially grows with the number of CAVs n or the planning horizon h (See Figure 3.1). The number of required MCTS iterations to sufficiently explore the tree of this size up to depth h also grows exponentially. This exponential growth severely limits the scalability of MCTS (Kurzer et al., 018a) for multi-agent problems including the CAV collision avoidance problem.

3.2.2 Coalition level approach

The second step of our hierarchical approach involves decision-making at the coalition level. Coalition leader receives a set of prioritized individual mitigation action plans $\{\alpha_h^i\}$ from each CAV $i \in \mathcal{C}$. These plans have been prioritized using the estimated Q^* values by the CAV level decision-making step. Each individual mitigation action plan α_h^i consists of h individual actions for CAV i. Coalition leader uses a simple beam search algorithm to select

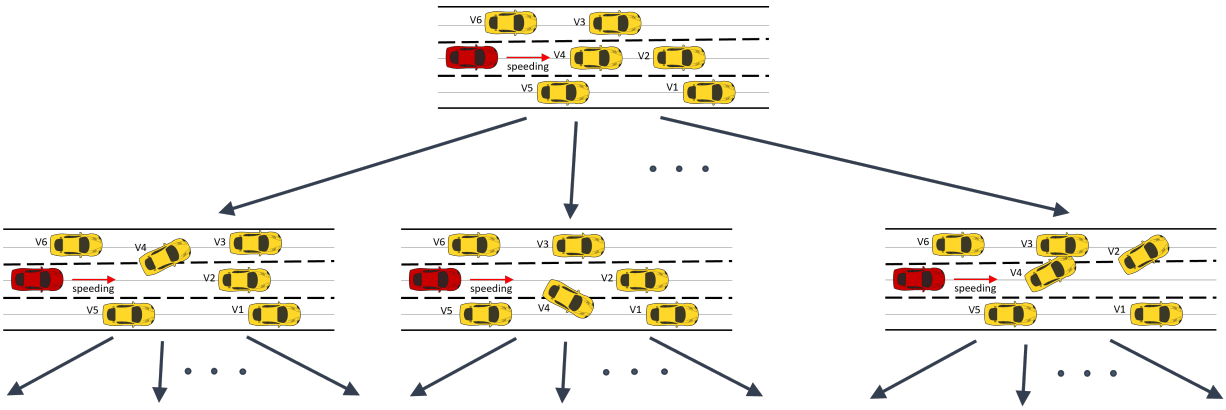


Figure 3.1: MCTS search tree for a coalition of 6 CAVs, the individual action space size $|A^i| = 5$ and the planning horizon $h = 3$. For regular MCTS, the branching factor equals 15625 and the size of the fully expanded tree $\approx 3.8 \times 10^{12}$. For COCOA, the branching factor equals 5 and the size of the fully expanded tree is 156.

a single mitigation action plan α_h^i for each CAV i, such that the collisions among the coalition members and the collisions with the misbehaving vehicle are avoided. Coalition leader sends back the selected mitigation action plan α_h^i to each CAV i.

3.3 Algorithms

To solve the proposed MMDP discussed in Section 3.2, we present four algorithms as part of the COCOA algorithms for a single coalition: Branching Factor Reduction, Intelligent Action Selection, Adaptive and Cooperative Reward Function, and Coalition level decision making using Beam Search. These algorithms overcome the limitations of MCTS such as limited scalability and reliability for the multi-agent problems by introducing significant changes to all four steps of the MCTS algorithm.

3.3.1 Branching Factor Reduction

We propose *branching factor reduction* method for MCTS to remove the exponential dependence of the MCTS tree size on the number of CAVs n and to improve the estimated Q^*

values. To this effect, we propose a novel design of the MCTS tree and modify selection, expansion, simulation, and backpropagation steps.

We call CAV i that is executing the MCTS algorithm, an ego CAV. In our novel MCTS tree structure, each tree node n at depth k corresponds to the coalition joint state s_{t_k}. Each tree edge, which is incoming to node n at depth k and is outgoing from node n_{parent} at depth $k-1$, corresponds to an individual action $a^i_{t_{k-1}}$ of the ego CAV i, as opposed to a joint action $a_{t_{k-1}}$ in the regular MCTS tree structure. This reduces the branching factor to $|A^i|$ as opposed to $|A^i|^n$ in the naive MCTS algorithm and gives us our desired scalability. The number of nodes in the tree with the branching factor $|A^i|$ and the tree depth h will have a total of $\frac{|A^i|^{(h+1)}-1}{|A^i|-1}$ possible nodes. For example, the tree in Figure 3.1 will only have 156 nodes with our new design as opposed to 3.8×10^{12} nodes in the regular MCTS tree structure.

We now describe an updated MCTS algorithm with the branching factor reduction method. A tree node n at depth k can be represented using a tuple (s_{t_k}, $a^i_{t_{k-1}}$, ν, cnt, n_{parent}, $N_{children}$), where s_{t_k} is the coalition joint state at time t_k, $a^i_{t_{k-1}}$ is the individual action for CAV i that corresponds to an action taken from n's parent at time t_{k-1}, ν is node value, cnt is node count, n_{parent} is the parent node of n, and $N_{children}$ stores the child nodes of n. At the beginning, ego CAV i initializes the search tree with a single root node n_μ with parameters ($s_{t_1}, a^i_{t_0} = null, \nu = 0, cnt = 0, n_{parent} = null, N_{children} = \{\}$) that stores the current coalition joint state at the current time t_1. Subsequently, ego CAV simulates many MCTS iterations that add more nodes to the tree. Each MCTS iteration consists of four steps (Browne et al., 2012) in this order: *Selection*, *Expansion*, *Simulation* and *Backpropagation*.

- *Selection*: In the selection step, ego CAV starts from the root node n_μ and select nodes that maximize UCT (upper confidence bounds for trees) value (Browne et al., 2012) until a leaf node is reached. UCT value strikes balance between exploration and

exploitation and is defined for j'th child of node n as below.

$$UCT = \bar{\nu_j} + C_M \sqrt{\frac{\log cnt}{cnt_j}} \tag{3.1}$$

where $\bar{\nu_j}$ is the average node value of j'th child, C_M is a constant, cnt is the node count of node n and cnt_j is the node count of j'th child.

When ego CAV i selects a node n at depth k from the parent node n_{parent} at depth $k-1$, the individual state $s^i_{t_k}$ for node n is determined using the individual action $a^i_{t_{k-1}}$ and the individual state $s^i_{t_{k-1}}$ at the parent node. However, ego CAV i needs individual actions $\{a^j_{t_{k-1}}, \forall j \in \mathcal{C} \backslash i\}$ for CAVs other than itself in order to determine states $\{s^j_{t_k}, \forall j \in \mathcal{C} \backslash i\}$ and assign a full joint state $s_{t_k} = s^i_{t_k} \cup \{s^j_{t_k}, \forall j \in \mathcal{C} \backslash i\}$ at the node n. To assign individual actions for CAVs other than itself, ego CAV i uses Intelligent Action Selection policy described in the next subsection.

- *Expansion*: In the expansion step, ego CAV expands the leaf node n_{leaf} by adding child nodes to the tree, one per each possible individual action a^i taken from state $n_{leaf}.s$.

- *Simulation*: In the simulation step, one of the newly expanded child nodes is selected. During the selection procedure of the child node, ego CAV i again uses the Intelligent Action Selection policy to assign individual actions for CAVs other than itself and determines the coalition joint state at the selected child node. Starting from this child node, ego CAV performs a Monte Carlo simulation by taking a random sample of the coalition joint action at each planning step until the planning horizon h or until a collision is detected.

- *Backpropagation*: In the above three steps, the reward is computed every time a new node is visited as well as at each step of the simulation phase. In the backpropagation step, the cumulative reward received at the end of the simulation is backpropagated to the root node in a reverse path to update node values and node counts. Node values

are increased by the reward amount and node counts are increased by one for nodes on this reverse path.

Since MCTS is an anytime algorithm, it can be stopped after either a fixed number of k iterations or by checking a convergence criterion on normalized node values. In the end, Q^* value can be estimated at each node n at depth k as:

$$Q^*(n_{parent}.s^i_{t_{k-1}}, n.a^i_{t_k}) \approx \frac{n.\nu}{n.cnt} \tag{3.2}$$

Where n_{parent} is the parent node of the node n. This value represents the estimate of the state-action value for the action a^i taken from the state $s^i_{t_{k-1}}$ at time t_{k-1}.

3.3.2 Intelligent Action Selection

We now describe the Intelligent Action Selection policy. This policy is used to determine individual actions for CAVs other than the ego CAV i during the selection and the simulation steps. When selection a node n from the node n_{parent}, we want to determine individual actions $\{a^j_{t_{k-1}}, \forall j \in \mathcal{C}\backslash i\}$ in order to fully determine the joint state s_{t_k} at the node n.

The simplest possible way to select actions for a CAV $j \in \mathcal{C}\backslash i$ is to select a random action from its action-space A^j. We used this simple approach in (Patel and Zalila-Wenkstern, 020a). It works fine for CAV coalitions with simple structures but has low reliability and takes comparatively more MCTS iterations to successfully avoid collisions in most scenarios.

We propose to use an Intelligent Action Selection policy (Patel and Zalila-Wenkstern, 021a) that makes significant improvements over the random action selection policy. In the Intelligent Action Selection policy, we use the statistics of the results received in the past MCTS iterations to perform action selection in the current MCTS iteration. The combination of the branching factor reduction method and the intelligent action selection policy constitutes the core COCOA CAV-level decision-making algorithm.

We define several additional parameters to store the result statistics of the past MCTS iterations at each MCTS node. A node n in MCTS search tree at depth k includes following parameters:

- $n.s_{t_k}$: the joint state of the coalition at time t_k
- $n.s^i_{t_k}$: the individual state of CAV i at time t_k
- $n.a^i_{t_{k-1}}$: the individual action for CAV i taken at time t_{k-1}
- $n.V$: a map indexed by $(j, a^j), \forall j \in \mathcal{C} \backslash i, a^j \in A^j$ that stores the sum of rewards received in all iterations that include node n while simulating action a^j for CAV j at time t_{k-1}
- $n.C$: a map indexed by $(j, a^j), \forall j \in \mathcal{C} \backslash i, a^j \in A^j$ that stores the number of iterations that included node n and in which action a^j was selected for CAV j at time t_{k-1}
- $n.\nu$: a value that corresponds to the sum of rewards of all iterations that include node n
- $n.cnt$: the visit count of node n
- $n.n_{parent}$: the parent node of node n
- $n.R_{temp}$: a map indexed by IDs of the coalition members and corresponds to the reward values received at this node by each coalition member during only the current MCTS iteration
- $n.N_{children}$ is the set of child nodes of node n

Ego CAV i initializes the root node n_μ of the search tree with the current joint state $n_\mu.s_{t_1}$, CAV i's current individual state $n_\mu.s^i_{t_1}$. Other variables $n_\mu.V$, $n_\mu.C$, $n_\mu.a^i_{t_0}$, $n_\mu.a_{t_0}$, $n_\mu.n_{parent}$ are set to null, $n_\mu.N_{children}$ is set to an empty set, and $n_\mu.\nu$ and $n_\mu.cnt$ are set to zero.

The most important functions of COCOA algorithm for CAV-level decision making executed by ego CAV i are outlined in Algorithm 1.

Algorithm 1 COCOA for a single CAV decision making

Require: n_μ, A
1: $k \leftarrow 1$
2: **while** maximum number of iterations are not executed **do**
3: $\quad \langle n_l, k, \mathcal{R} \rangle \leftarrow$ COCOA-SelectionPolicy(n_μ, k, A)
4: $\quad$ **if** $n_l.cnt \neq 0$ **then**
5: $\quad\quad \langle n_l, k, \mathcal{R} \rangle \leftarrow$ COCOA-ExpansionPolicy$(n_l, k, A, \mathcal{R})$
6: $\quad$ **end if**
7: $\quad \mathcal{R} \leftarrow$ COCOA-SimulationPolicy$(n_l, k, \mathcal{R})$
8: $\quad$ COCOA-BackpropagationPolicy$(n_{l,k,\mathcal{R}})$
9: **end while**

We now describe all four steps of the MCTS algorithm modified to work with the novel tree structure for the intelligent action selection and the branching factor reduction.

COCOA selection policy

Starting from the root node n_μ, ego CAV i consecutively selects nodes in each step using UCT algorithm (Browne et al., 2012) applied over $N_{children}$ using node statistics ν and cnt in step 4 and 5 in Algorithm 2. The child node n that is selected automatically determines action $a^i_{t_k}$ for the ego CAV i. To select actions $a^j_{t_k}$ of other coalition members $j \in \mathcal{C}\backslash i$, ego CAV i applies UCT algorithm over the stored action reward values $V(j)$ and the count values $C(j)$ in step 7 in Algorithm 2. UCT value for action selection of a coalition members $j \in \mathcal{C}\backslash i$ is defined as below.

$$UCT(a^j) = \frac{n.V(j, a^j)}{n.C(j, a^j)} + C_A \sqrt{\frac{\log n.cnt}{n.C(j, a^j)}} \tag{3.3}$$

$a^i_{t_k}$ combined with $a^{\mathcal{C}\backslash i}_{t_k}$ forms the joint action set a_{t_k}, which is then applied to the joint state set s_{t_k} of the parent node n_{parent} to derive n's joint state set $s_{t_{k+1}}$ in step 9 in Algorithm 2. Separate reward values for all coalition members are computed and stored in R_{temp} array. The values in R_{temp} are then used to update the cumulative reward $\mathcal{R}$.

Algorithm 2 COCOA-Selection Policy

Require: n_μ, k, A
Ensure: $n, k, \mathcal{R}$
1: $n \leftarrow n_\mu$
2: $\mathcal{R} \leftarrow 0$
3: **repeat**
4: $a^i_{t_k} \leftarrow \text{UCTAction}(A^i, n.N_{children})$
5: $n \leftarrow$ node with $a^i_{t_k}$ action from $n.N_{children}$
6: **for** j in $\mathcal{C} \backslash i$ **do**
7: $n.a^j_{t_k} \leftarrow \text{UCTAction}(A^j, n.V(j), n.C(j))$
8: **end for**
9: $n.s_{t_{k+1}} \leftarrow \text{ComputeState}(n_{parent}.s_{t_k}, n.a_{t_k})$
10: $n.R_{temp} \leftarrow \text{ComputeReward}(n_{parent}.s_{t_k}, n.s_{t_{k+1}}, n.a_{t_k})$
11: $\mathcal{R} \leftarrow \mathcal{R} + n.R_{temp}(i)$
12: **for** j in $\mathcal{C} \backslash i$ **do**
13: $\mathcal{R} \leftarrow \mathcal{R} + \lambda \cdot n.R_{temp}(j)$
14: **end for**
15: $k \leftarrow k + 1$
16: **until** n is a leaf node

COCOA expansion policy

To expand a node n in the tree, ego CAV i first computes new individual states $s^i_{t_{k+1}}$ using the current individual state $s^i_{t_k}$ and each valid individual CAV action $a^i_{t_k}$. The validity of actions is determined using the preconditions listed in Table 3.1. For an individual action space size of 5, up to five child nodes are added in $N_{children}$. The task of computation of joint states $s_{t_{k+1}}$ for these child nodes as well as the selection task of the child node is left to *COCOA-SelectionPolicy* call in step 6 in Algorithm 3, which uses UCB1 algorithm to select the actions for CAVs other than the ego CAV and computes the corresponding joint states.

COCOA Simulation policy

Ego CAV performs the simulation step starting from a leaf node n_l of the MCTS tree. The joint action set a_{t_k} is selected randomly from the joint action space A. The next state $s_{t_{k+1}}$ is derived from the current node's joint state s_{t_k} and the sampled joint action a_{t_k}. Using

Algorithm 3 COCOA-Expansion Policy

Require: $n, k, A, \mathcal{R}$
Ensure: $n_l, k, \mathcal{R}$
1: **for** $a^i_{t_k}$ in A^i **do**
2: $s^i_{t_{k+1}} \leftarrow$ computeState$(n.s^i_{t_k}, a_{t_k})$
3: $n_{child} \leftarrow$createNode$(a^i_{t_k}, s^i_{t_{k+1}})$
4: $n.N_{children} \leftarrow n.N_{children} \cup n_{child}$
5: **end for**
6: $\langle n_l, k, \mathcal{R} \rangle \leftarrow$ COCOA-SelectionPolicy(n, k, A)

the derived joint state, a new child node n_{child} is created. Separate reward values for all coalition members are computed and stored in R_{temp} array which is later used to update the cumulative reward $\mathcal{R}$. Here, note that n_{child} is not added to the MCTS tree. This procedure is performed until a terminal node is found. A node is considered terminal when a collision among the coalition members is detected or the planning horizon h is reached.

Algorithm 4 COCOA-Simulation Policy

Require: $n, k, \mathcal{R}$
Ensure: $\mathcal{R}$
1: **repeat**
2: $a_{t_k} \leftarrow$ RandomSelection(A)
3: $s_{t_{k+1}} \leftarrow$ ComputeState$(n.s_{t_k}, a_{t_k})$
4: $n_{child} \leftarrow$ ExpandSingleChildNode(n, a_{t_k})
5: $R_{temp} \leftarrow$ ComputeReward$(n.s_{t_k}, n_{child}.s_{t_{k+1}},$
$n_{child}.a_{t_k})$
6: $\mathcal{R} \leftarrow \mathcal{R} + R_{temp}(i)$
7: **for** j in $\mathcal{C} \backslash i$ **do**
8: $\mathcal{R} \leftarrow \mathcal{R} + \lambda \cdot R_{temp}(j)$
9: **end for**
10: $n \leftarrow n_{child}$
11: $k \leftarrow k + 1$
12: **until** n is a terminal node

COCOA Backpropagation policy

During the backpropagation step, the cumulative reward computed at the end of the simulation step is used to update $n.\nu$ and $n.cnt$ values at each node in the reverse path from

the leaf to the root. Values in V and C are also updated using the values stored in R_temp for the actions that were selected during the current iteration for CAVs other than the ego CAV. Additionally, all the joint state values s as well as the action values of the CAVs other than the ego CAV $a_{t_k}^{\mathcal{C}\backslash i}$ for all the nodes along the reverse path are set back to *null* in their respective parameter vectors except at the root node. At the end of the algorithm execution,

Algorithm 5 COCOA-Backpropagation Policy

Require: $n, k, \mathcal{R}$
while $n \neq n_\mu$ **do**
  $n.cnt \leftarrow n.cnt + 1$
  $n.\nu \leftarrow n.\nu + \mathcal{R}$
  for j in $\mathcal{C}\backslash i$ **do**
    $n.V(j, n.a_{t_k}^j) \leftarrow n.V(j, n.a_{t_k}^j) + n.R_{temp}(j)$
    $n.C(j, n.a_{t_k}^j) \leftarrow n.C(j, n.a_{t_k}^j) + 1$
  end for
  $n.s_{t_k} \leftarrow$ null
  $n.a_{t_k}^{\mathcal{C}\backslash i} \leftarrow$ null
  $n \leftarrow n.n_{parent}$
  $k \leftarrow k - 1$
end while

ego CAV i estimates the Q^* values using Equation 3.2 for each node of the MCTS tree. Each path in the MCTS tree from the root node to a leaf node represents one mitigation action plan. MCTS tree is stripped of extra variables. Only individual states, individual actions, and estimated Q^* values are retained at each node of the tree. This tree is then shared with the coalition leader using V2V communication.

3.3.3 Adaptive and Cooperative Reward Function

We consider several reward factors that affect CAV's safety, efficiency and comfort in the reward function used by the COCOA algorithm. Ego CAV i's reward $r_{t_k}^i$ at time t_k is defined as a dot product of two vectors: a weight vector $\overline{w}$ and a reward factors vector $\overline{r}^i$.

$$r_{t_k}^i = \overline{w} \cdot \overline{r}^i \tag{3.4}$$

which is equivalent to,

$$r^i_{t_k} = \sum_j w_j r^i_j$$

where $\overline{r}^i = \{r^i_j\}$ is a vector consisting of several reward factors indexed by j as follows:

- r^i_{ccol}: *coalition collision* corresponds to the boolean value indicating if there is a collision between CAV i and another coalition member while simulating an action $a^i_{t_k}$ during the current MCTS iteration
- r^i_{mcol}: *misbehaving collision* corresponds to the boolean value indicating if there is a collision between CAV i and the misbehaving vehicle while simulating an action $a^i_{t_k}$ during the current MCTS iteration
- r^i_{accel}: Current action $a^i_{t_k}$'s acceleration
- r^i_{decel}: Current action $a^i_{t_k}$'s deceleration
- r^i_{sd}: *speed deviation* corresponds to the difference of the current speed $s^i_{t_k}.speed$ and the initial speed $s^i_{t_1}.speed$ of the CAV
- r^i_{lc}: *lane change* corresponds to the difference between the current lane index $s^i_{t_k}.lane$ and the previous lane index $s^i_{t_{k-1}}.lane$
- r^i_{stop}: *full stop* corresponds to the boolean value indicating if the current speed $s^i_{t_k}.speed$ equals zero
- r^i_{mv}: *max velocity* corresponds to the boolean value indicating if the current speed $s^i_{t_k}.speed$ goes above the allowed maximum velocity of the vehicle

In Equation 3.4, the weight vector $\overline{w}$ consists of weight values $\{w_j\}$, each of which is multiplied with exactly one reward factor from the set of reward factors. Each of the reward

factors represents one or more action. For example, r^i_{accel} represents *accel* action and r^i_{lc} represents *cll* and *clr* actions. Setting a high or low value for a particular weight w_j makes ego CAV i to eventually either prioritize or deprioritize the multiplied reward factor's represented actions in its final set of prioritized action plans. For example, if we set a relatively high weight value w_{accel} which is multiplied with the reward factor r^i_{accel}, then CAV i prioritizes action *accel* higher than other actions in its final action plans. An important observation is that some actions should be prioritized in a particular coalition state but in a completely different coalition state, the same of set of actions may lead to a collision among coalition members. Using the same set of reward weights for all actions at each tree node in an MCTS iteration is suboptimal, as different tree nodes represent different coalition states. Below we present a novel coalition state-based reward weights updating algorithm for the adaptive reward function.

Coalition state based reward weights update When simulating an MCTS iteration, ego CAV i executes a sequence of actions $\{a^i_{t_k}\}^h_{k=1}$. When simulating each of the actions in $\{a^i_{t_k}\}^h_{k=1}$ sequence, coalition states $s^i_{t_k}$ will be different at different depth values k. For instance, consider the coalition state at time t_k in Figure 3.2. Consider two actions for the ego

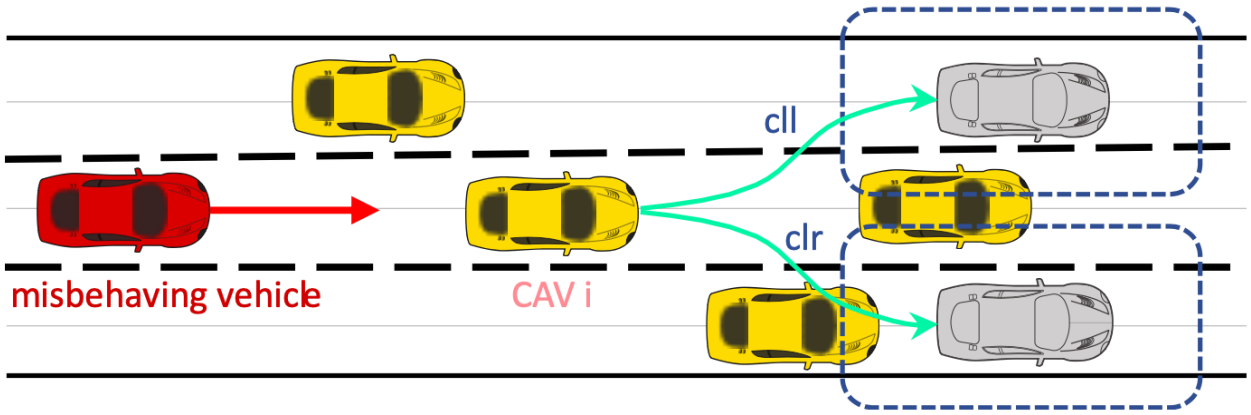


Figure 3.2: *cll* and *clr* actions for CAV i at time t_k

CAV at time step t_k: *change lane to left (cll)* and *change lane to right (clr)*. The resultant

positions of the ego CAV at time step t_{k+1} for both actions are shown in Figure 3.2. We consider the neighborhood of each of these positions. There is one coalition member in the neighborhood for action *cll*'s resultant position, whereas there are two coalition members in the neighborhood for action *clr*'s resultant position. While the ego CAV selects and simulates its chosen action during the current MCTS iteration, it will also select and simulate actions for the neighborhood CAVs simultaneously in the same MCTS iteration. This may lead to a collision if the selected actions of the neighborhood CAVs are in conflict with the ego CAV's selected action. If we consider all future actions that can be selected for the neighborhood CAVs and take all the action combinations of the neighborhood CAVs, then the proportion of neighborhood CAVs' action combinations that can have a collision with the ego CAV are greater if the ego CAV selects the action *clr* than if the ego CAV selects the action *cll*. Thus more penalty (or less reward) should be received for choosing the action *clr* than the action *cll*. We achieve this by updating the reward weights dynamically before simulating each action during the MCTS iteration. We assume that the weights are negative values and represent the penalty. Reward weights are temporarily modified for each action $a^i_{t_k}$ as below:

$$\overline{w} = (1 + \beta\mathcal{N}^i)\overline{w} \tag{3.5}$$

where

- $\mathcal{N}^i$ is the number of vehicles in CAV i's neighborhood after simulating an action $a^i_{t_k}$ during the current MCTS iteration
- β is the coefficient in range $[0, 1]$

During the simulation of each MCTS iteration, ego CAV uses the modified reward weights as given in Equation 3.5 for each action until the planning horizon. During the backpropagation phase of the MCTS iteration, reward values computed using Equation 3.4 are used to update each visited node's parameters in the MCTS tree in reverse order from the leaf node to the root node.

3.3.4 Coalition level decision making using Beam Search

The coalition leader performs the action selection for CAVs using the *beam-search* method beamsearch. Beam search with the beam size = 1 is employed to ensure that the best possible sequence of actions for each coalition member satisfy the collision constraints, when possible.

Algorithm 6 COCOA Beam Search for coalition leader

Require: $T_i, \forall i \in \mathcal{C}$
Ensure: $\alpha_h^i, \forall i \in \mathcal{C}$
1: Initialize $currentNodes$, $sortedChildNodes$ to null maps
2: **for** i in $\mathcal{C}$ **do**
3: $\quad currentNodes(i) \leftarrow T_i.n_\mu$
4: **end for**
5: $k \leftarrow 1$
6: **while** $k \leq h$ **do**
7: $\quad$ **for** i in $\mathcal{C}$ **do**
8: $\quad\quad sortedChildren(i) \leftarrow$ sortByQValues$(currentNodes(i).N_{children})$
9: $\quad$ **end for**
10: $\quad nonCollidingNodes \leftarrow$ RecCombinationSelection$(currentNodes, sortedChildren)$
11: $\quad$ **if** $nonCollidingNodes =$ null **then**
12: $\quad\quad$ No solution is found.
13: $\quad$ **end if**
14: $\quad$ **for** i in $\mathcal{C}$ **do**
15: $\quad\quad \alpha_h^i \leftarrow \alpha_h^i \cup nonCollidingNodes(i).a_{t_k}^i$
16: $\quad$ **end for**
17: $\quad currentNodes \leftarrow nonCollidingNodes$
18: **end while**

Initially, at the tree depth k=1, we store root nodes for each CAV's tree as that CAV's current node under consideration in line 3 of Algorithm 6. The goal is to select one child node per CAV from the children of each CAV's current node, such that the joint combination of selected nodes for all CAVs is non-colliding.

At each value of k from 1 to the planning horizon h, we take child nodes of each CAV's current node and sort these child nodes according to their Q^* values in line 8. From the list of sorted child nodes of each CAV, we select one node per CAV using the Recursive

Combination Selection algorithm such that the combination of all CAV's selected nodes is non-colliding in line 10. In line 15, we update each CAV's action plan by adding a new action from that CAV's selected non-colliding node. In line 17, we update each CAV's current node to be the selected non-colliding node and repeat these steps for the next value of k. At the end, we will have generated a single action plan α_h^i for each CAV i, such that the combination of action plans of different CAVs is non-colliding.

We now describe the recursive combination selection algorithm. This algorithm takes a list of sorted child nodes for each CAV as an input and outputs one child node per CAV such that the combination of selected child nodes for all CAVs is non-colliding, when possible. It tries all possible combinations of sorted child nodes in the descending order of Q^* values using a novel recursive algorithm. It first picks one CAV that has more than one child node in its list in lines 2-6. If all CAVs only have one child node, then it checks if the joint combination is non-colliding in line 8. If the join combination is non-colliding, then it is set as the final result, otherwise, the final result is set as null in lines 8-15. If there is at least one CAV (say CAV i) with more than one child node, then we split the list of child nodes for this CAV i into two sets: one set of a single child, and another set of remaining child nodes (see lines 16-17). First, we select a set of a single child node for CAV i and merge it with sets of child nodes of other CAVs to create a new map $newChildNodes1$. We call the same recursive combination selection algorithm on this new map $newChildNodes1$ to see if we find a solution in line 19. If the solution is not found, then in line 24, we call the same recursive combination selection algorithm on the map $newChildNodes2$, which is generated by merging the set of remaining child nodes of CAV i with the sets of child nodes of other CAVs. If the solution is found by this recursive call, then it is set as the final result in line 26, otherwise, the final result is set as null.

At the end of the beam search algorithm (Algorithm 6) execution, the coalition leader generates individual action plans for each CAV. It sends these plans to the respective CAVs

Algorithm 7 Recursive combination selection for Beam Search

Require: $currentNodes$, $sortedChildNodes$
Ensure: $nonCollidingCombination$

$i \leftarrow 0$
for i in $\mathcal{C}$ **do**
    if $currentNodes(i).N_{children}.size > 1$ **then**
        $nonSingleCAV \leftarrow i$
        break loop
    end if
end for
if $nonSingleCAV =$ null **then**
    $result \leftarrow$ checkCollision($currentNodes$)
    if $result =$ False **then**
        $nonCollidingCombination = currentNodes$
    else
        $nonCollidingCombination =$ null
    end if
else
    $singleChild \leftarrow sortedChildNodes(i)[0]$
    $otherChildren \leftarrow sortedChildNodes(i)[1, |A^i|]$
    $newChildNodes1 \leftarrow singleChild \cup newChildNodes(\mathcal{C} \backslash i)$
    $result1 \leftarrow$ RecCombinationSelection($currentNodes, newChildNodes1$)
    if $result1 \neq$ null **then**
        $nonCollidingCombination = result1$
    else
        $newChildNodes2 \leftarrow otherChildren \cup newChildNodes(\mathcal{C} \backslash i)$
        $result2 \leftarrow$ RecCombinationSelection($currentNodes, newChildNodes2$)
        if $result2 \neq$ null **then**
            $nonCollidingCombination = result2$
        else
            $nonCollidingCombination = null$
        end if
    end if
end if

using V2V communication. Each CAV executes the action in the received action plans to achieve collision avoidance.

CHAPTER 4

CASE STUDY FOR SINGLE COALITION ALGORITHMS

In order to examine the behavior of our proposed COCOA algorithm in real-world scenarios involving a single coalition of several CAVs, we perform several simulations in MATISSE [1]. CAV agents navigate in a coalition formation on a three-lane highway. In the scenarios, we consider different types of misbehaviors at different positions of the misbehaving vehicle with respect to the coalition.

We use c to represent a MATISSE simulation cycle which corresponds to the execution of the *agent cycle* and the *environment cycle* (Torabi et al., 2018). In the agent cycle, the agents 1) receive the latest environment state (i.e., perception) and communications from other agents, 2) execute their decision-making algorithms, and 3) send their intended actions to the environment component and messages (i.e., communication) to other agents. In the environment cycle, the environment component combines the agent actions and applies the physical laws to determine the new agent states and the environment state.

In our case, the execution of Algorithm 1 is performed during the agent cycle. A vehicle's decision-making process in case of detection/warning of a misbehaving vehicle executes iterations of COCOA algorithm for the processing time of up to 2 seconds. Each CAV plans its actions for the planning horizon of 6. A time step t corresponds to one simulation cycle. We use Δt=10.

4.1 Case Study 1: Acceleration misbehavior

As depicted in Figure 4.1, Scenario 1 considers a coalition of 6 vehicles, led by vehicle $V6$. Each of the vehicles in the coalition is navigating at the cruising speed of 2 units/cycle. At cycle c_n, vehicle $V4$ detects the misbehaving vehicle, coming from behind at the speed of 5

[1] Demos available at www.utdallas.edu/∼dhruv/CAVdemos

units/cycle. $V4$ immediately sends the estimated state parameters including position and speed of the misbehaving vehicle to other coalition members. At cycle c_{n+1}, after receiving the misbehaving alert, each coalition member executes Algorithm 1 $COCOA(n_\mu, A)$ to derive their individual MCTS trees and to estimate state-action values Q^* for actions in their individual action space. The root node n_μ of CAV i's MCTS tree contains the coalition current joint state, i.e., the current states of all coalition members. The joint action space A contains possible actions for each coalition member for each possible state up to the planning horizon. A COCOA iteration's expansion step, i.e., $COCOA-ExpansionPolicy(n, k, A, \mathcal{R})$ in Algorithm 3, only considers the actions that ego CAV i can perform. For instance, consider CAV $V3$ as our ego CAV. In Figure 4.1 , there are four possible actions for $V3$: *accel*, *decel*, *maintain* and *clr*, and four child nodes are added to $V3$'s root node during COCOA's single iteration execution performed by $V3$. *cll* is not available for $V3$ as it will drive the CAV out of the road area (see Table 3.1 for the list of all CAV actions). Actions for coalition members other than $V3$ are sampled using the Intelligent Action Selection policy (step 7 of Algorithm 2) as well as in the COCOA expansion policy (step 6 of Algorithm 3). Whereas in COCOA simulation step (step 2 of Algorithm 4), actions for all coalition members including $V3$ are randomly sampled. After $COCOA$ in Algorithm 1 is executed, each vehicle shares its derived MCTS tree with the leader $V1$. At cycle c_{n+2}, $V1$ receives the MCTS trees from all vehicles in the coalition, and derives final action plans for each coalition member. Table 4.1 shows the final actions chosen by $V1$ for each coalition member in Case Study 1. At cycle c_{n+3}, CAVs start executing their final action plans to avoid collisions with the misbehaving vehicle. Figure 4.1 shows the actions taken by each coalition member. In Figure 4.1, the first subfigure shows the initial coalition state. Each subsequent subfigure shows the actions taken by each CAV in the coalition. For the first action at cycle c_{n+3}, $V4$ changes its lane to the first lane in order to avoid the collision with the misbehaving vehicle. $V2$ accelerates to gain safe speed and avoid future collisions with the misbehaving vehicle. At

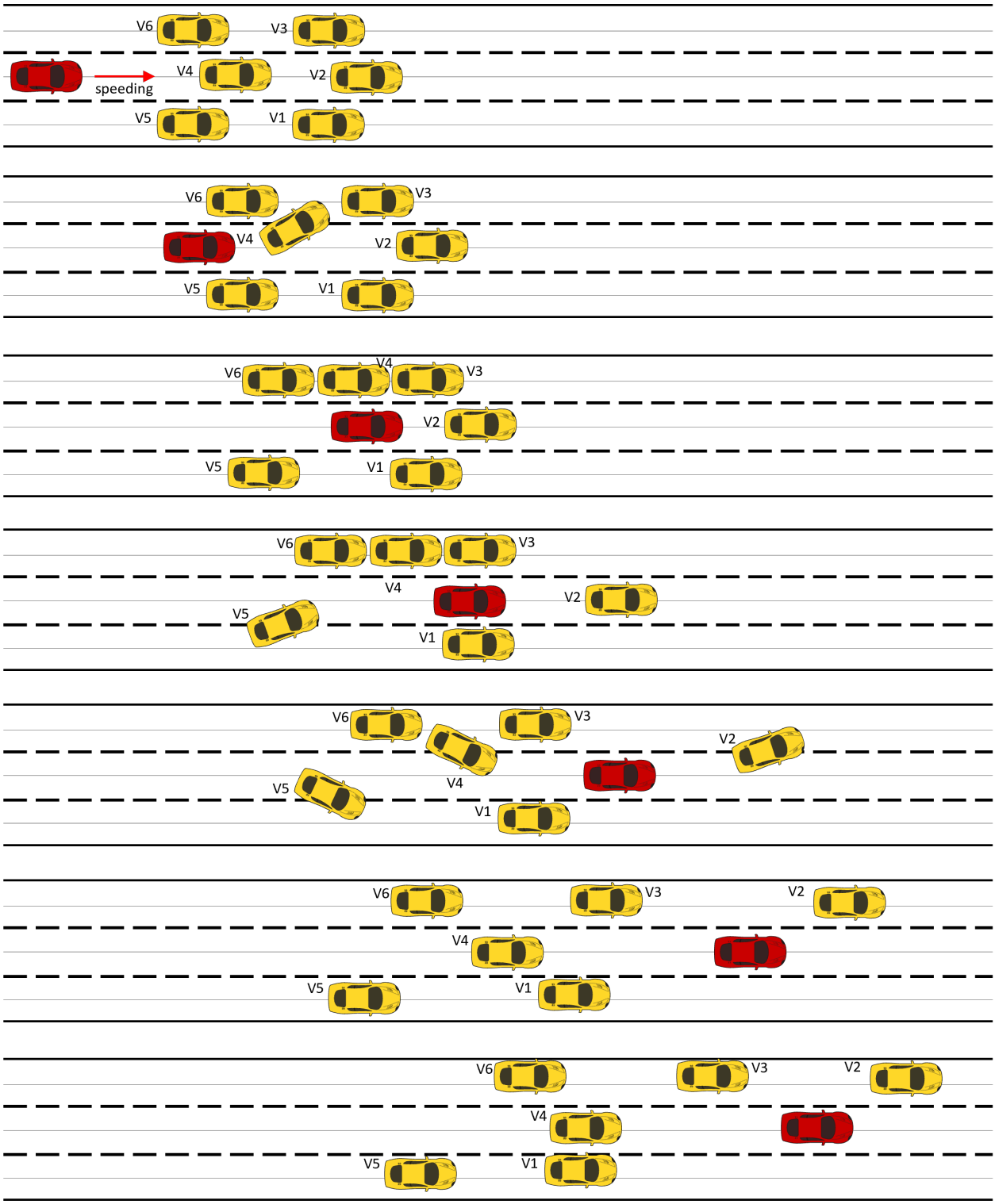


Figure 4.1: Case Study 1: Acceleration misbehavior type

Table 4.1: Acceleration misbehavior: Final actions chosen by coalition leader

CAV i	Final Action Plan α_h^i
V_1	*maintain*, *maintain*, *maintain*, *maintain*, *decel*, *accel*
V_2	*accel*, *maintain*, *maintain*, *cll*, *decel*, *accel*
V_3	*maintain*, *maintain*, *maintain*, *accel*, *maintain*, *accel*
V_4	*cll*, *maintain*, *maintain*, *clr*, *maintain*, *accel*
V_5	*maintain*, *decel*, *cll*, *clr*, *maintain*, *accel*
V_6	*maintain*, *maintain*, *maintain*, *maintain*, *maintain*, *accel*

cycle c_{n+13}, $V5$ decelerates and then change its lane to the second lane at cycle c_{n+23}. Since $V4$ is sandwiched between $V3$ and $V4$, it changes its lane to the second lane at cycle c_{n+33}. $V5$ starts accelerating to allow a safe lane change maneuver to $V4$ at the same cycle c_{n+33}. $V5$ changes its lane back to the third lane at the same cycle c_{n+33} as well. These seemingly unnecessary actions by $V5$, i.e. changing its lane to the left lane at the cycle c_{n+23} and changing its lane back to the third lane at the cycle c_{n+33}, are performed as they have higher Q^* values in the MCTS tree of and $V5$. Last two actions by all CAVs at the cycles c_{n+43} and c_{n+53} mostly seem unnecessary except for $V2$ and $V3$. They are chosen for the same reason that the chosen actions have a high value of Q^* in the respective MCTS trees of the CAVs.

4.2 Case Study 2: Break misbehavior

In this scenario, we consider a coalition of 6 CAVs in the same dense formation as in Case Study 1. The misbehaving vehicle is detected to be breaking in front of the coalition by vehicle $V3$ at cycle c_n. Table 4.2 shows the final actions chosen by the leader $V1$ for each coalition member in Case Study 1. Two simulation cycles are used up by the coalition

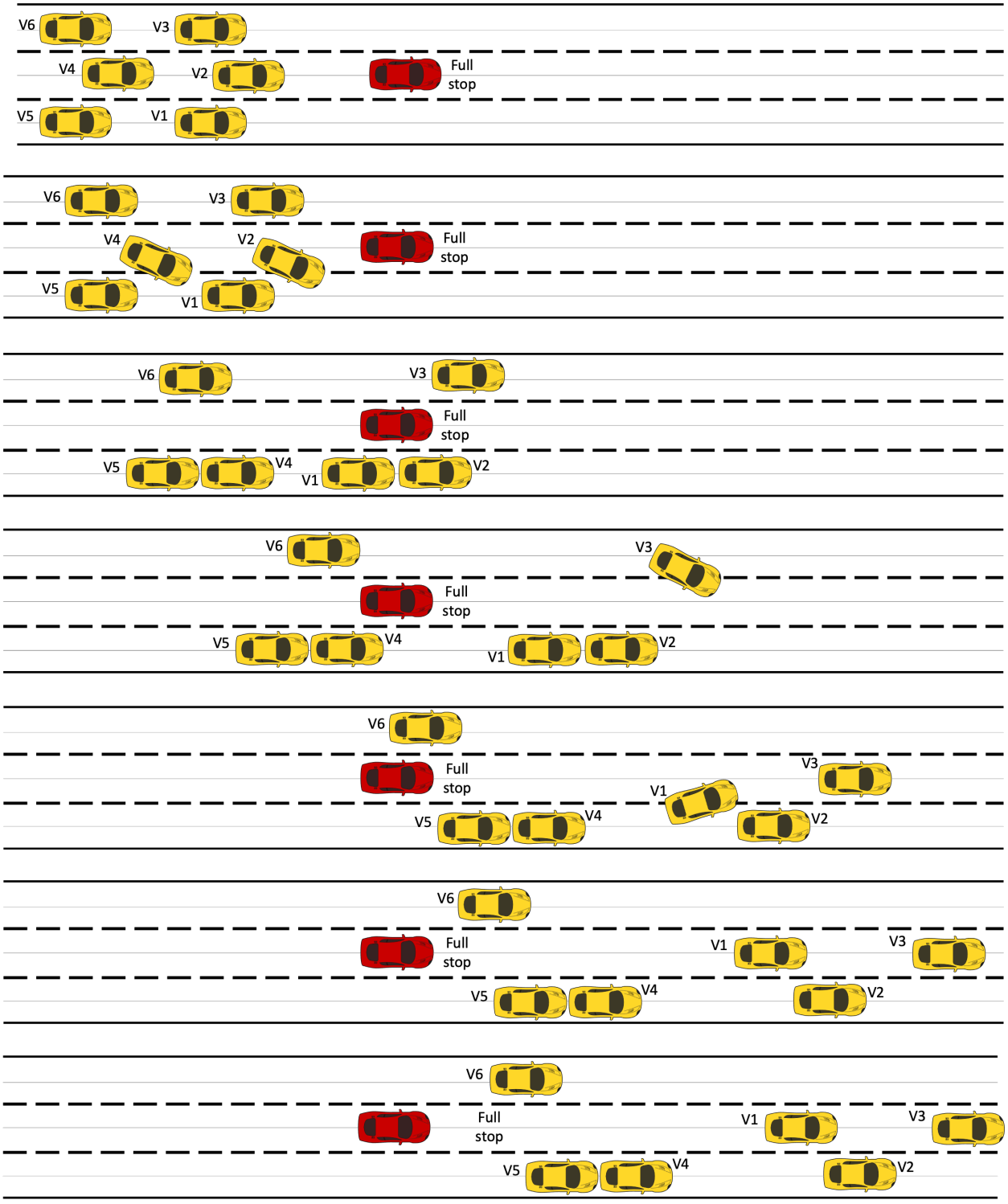


Figure 4.2: Case Study 2: Break misbehavior type

Table 4.2: Break misbehavior: Final actions chosen by coalition leader

CAV i	Final Action Plan α_h^i
V_1	*maintain*, *accel*, *decel*, *cll*, *decel*, *accel*
V_2	*clr*, *accel*, *decel*, *maintain*, *decel*, *accel*
V_3	*accel*, *maintain*, *clr*, *maintain*, *maintain*, *accel*
V_4	*clr*, *maintain*, *accel*, *decel*, *decel*, *accel*
V_5	*maintain*, *maintain*, *accel*, *decel*, *decel*, *accel*
V_6	*maintain*, *maintain*, *maintain*, *maintain*, *maintain*, *accel*

members in their collaborative decision-making task. Starting from the third cycle c_{n+3}, coalition members start executing their final action plans derived by the leader $V1$. At the cycle c_{n+3}, $V2$ immediately changes its lane to the third lane in order to avoid the collision with the misbehaving vehicle. Similarly, $V4$ also changes its lane to the third lane at cycle c_{n+3} to avoid the future collision with the misbehaving vehicle. Note that $V1$ and $V2$ are now very close in the same lane and also very close to the misbehaving vehicle on its right lane. At cycle c_{n+13}, $V1$ and $V2$ both perform acceleration actions in order to pass the misbehaving vehicle and thus to be out of the collision danger, while other coalition members maintain their speeds. At cycle c_{n+23}, $V1$ and $V2$ both decelerate in order to gain their normal speeds. At the same cycle c_{n+23}, $V3$ decides to change its lane to the second lane after the misbehaving vehicle during its previous action. Since $V1$ and $V2$ are very close, $V1$ decides to change its lane to the second lane at the cycle c_{n+33}. At cycles c_{n+43} and c_{n+53}, no CAVs change their lanes anymore and continue on the straight paths by maintaining their speeds, accelerating or decelerating.

Table 4.3: Zigzag misbehavior: Final actions chosen by coalition leader

CAV i	Final Action Plan α_h^i
V_1	*accel*, *maintain*, *maintain*, *maintain*, *maintain*, *decel*
V_2	*accel*, *maintain*, *maintain*, *decel*, *decel*, *accel*
V_3	*accel*, *maintain*, *maintain*, *maintain*, *decel*, *maintain*
V_4	*clr*, *maintain*, *cll*, *clr*, *maintain*, *accel*
V_5	*maintain*, *decel*, *maintain*, *cll*, *accel*, *cll*
V_6	*accel*, *maintain*, *maintain*, *maintain*, *maintain*, *decel*

4.3 Case Study 3: Zigzag misbehavior

In this scenario as well, we consider the same coalition structure of 6 CAVs in the dense formation as in Case Studies 1 and 2. CAV $V4$ detects the misbehaving vehicle accelerating from behind. The misbehaving vehicle is traveling at a higher speed of 5 units/cycle on a zigzag path instead of a straight path as in Case Study 1.

After $V4$ detects the misbehaving vehicle coming at the speed of 5 units/cycle on a zigzag path, it immediately changes its lane to the third lane at cycle c_{n+3} in order to avoid the emergent collision with the misbehaving vehicle. In order to allow this lane change by $V4$, $V1$ decides to accelerate at the cycle c_{n+3} in order to give enough space to $V2$. At the same cycle c_{n+3}, $V2$, $V3$, and $V6$ also decide to accelerate in order to avoid future collisions with the misbehaving vehicle. Note that the misbehaving vehicle is navigating on a zigzag path between the first lane and the second lane. The vehicles that decide to accelerate, i.e., $V2$, $V3$, and $V6$ are also the vehicles that were navigating on either the first or the second lane. For the next action at the cycle c_{n+13}, all vehicles decide to maintain their speeds. Only

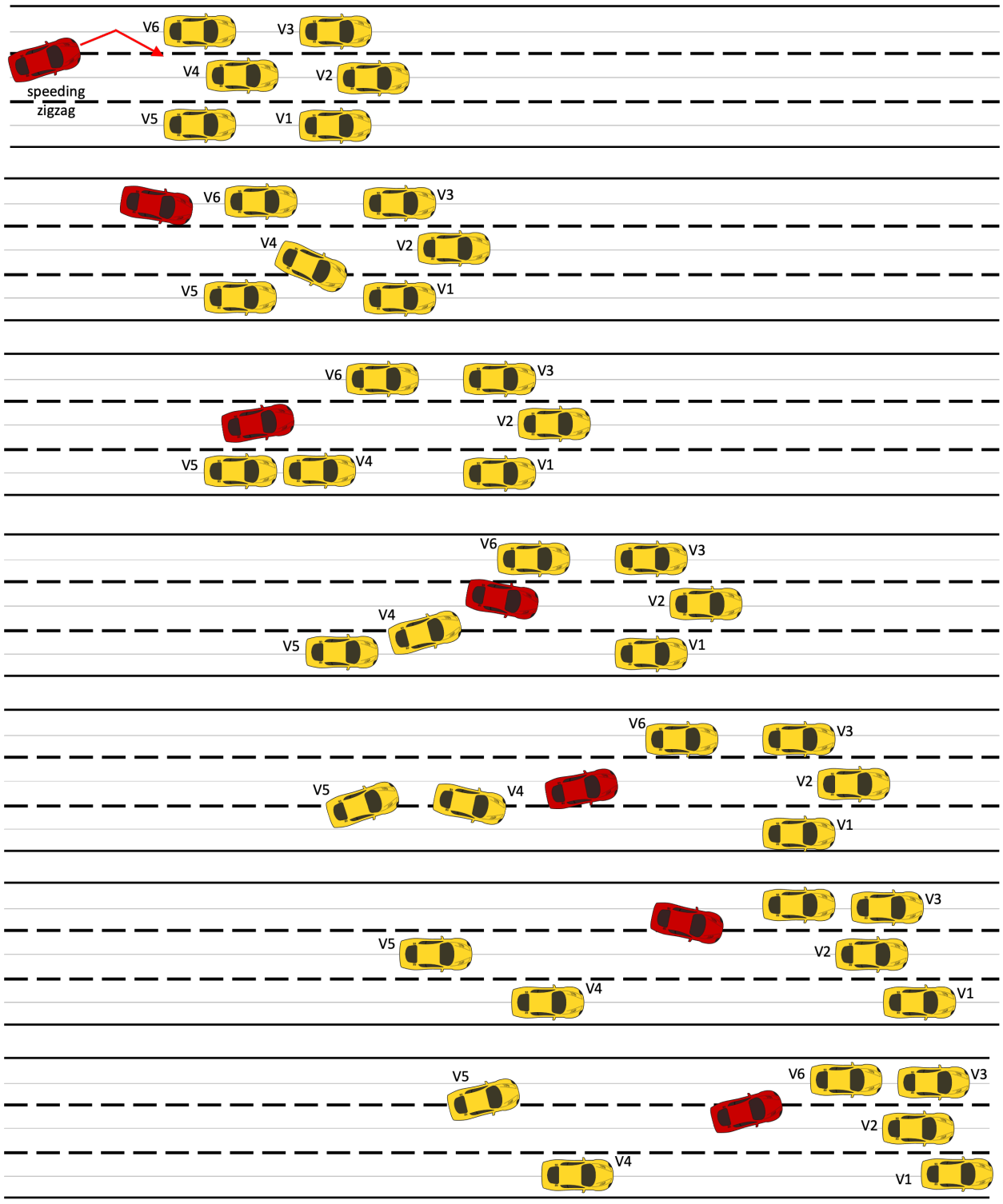


Figure 4.3: Case Study 3: Zigzag misbehavior type

CAV $V5$ decides to decelerate at c_{n+13} as it's too close to $V4$. After the misbehaving vehicle has passed the CAV $V4$, $V4$ decides to change its lane back to the second lane at the cycle c_{n+23}, while other CAVs maintain their speeds. At cycles c_{n+33}, $V4$ changes its lane to the third lane while $V5$ changes its lane to the second lane. At the cycle c_{n+43}, $V5$ further makes a lane change to the first lane. These seemingly unnecessary actions by $V4$ and $V5$ are due to high Q^* values for the chosen actions in their MCTS trees computed by the COCOA algorithm. $V1$, $V2$, $V3$, and $V6$ maintain their lanes and mostly maintain their speeds in order to keep ahead of the misbehaving vehicle coming behind them in order to avoid any collisions with the misbehaving vehicle.

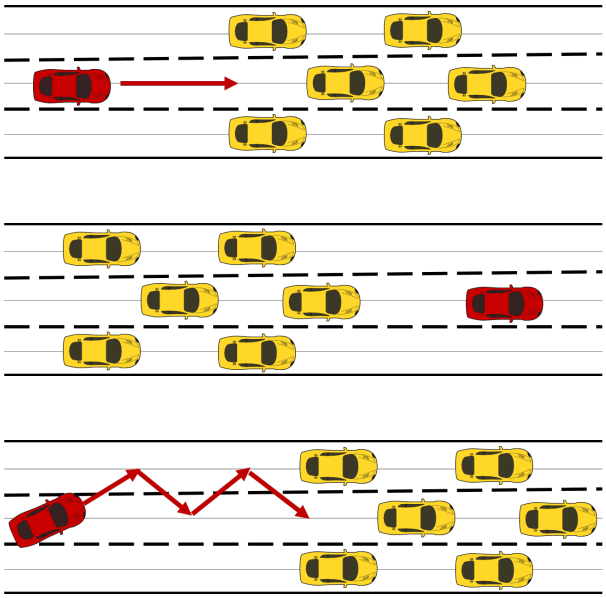
CHAPTER 5

SINGLE COALITION ALGORITHMS EVALUATION USING SIMULATION EXPERIMENTS

In this chapter, we evaluate COCOA algorithm by implementing it in MATISSE, a multi-agent-based traffic simulation system. We obtain experimental simulation data and use it to choose values of several design parameters used in COCOA. Additionally, we implement the state-of-the-art centralized (Nakamura et al., 2020) and decentralized algorithms (Kurzer et al., 018b) in MATISSE and compare the reliability and the scalability with COCOA. We first discuss the multi-agent-based traffic simulation system, then present the experimental settings used in the parameter tuning experiments, the reliability experiments, and the scalability experiments.

5.1 Simulation Experimental Setting

We have implemented COCOA, DeCoC-MCTS (Kurzer et al., 018b) and the Integer Programming (IP) centralized algorithm (Nakamura et al., 2020) in MATISSE. In the virtual MATISSE simulation environment, the unit of time is called *cycle* and the unit of length is simply called *unit*. In our simulation experiments, a coalition of CAVs is navigating on a three-lane highway at the speed of 2 units/cycle. During the simulation experiment, we add a misbehaving virtual CAV to the simulation that misbehaves to cause a potential collision with one or more coalition CAVs. Before the collision occurs, one of the affected coalition CAVs detects the misbehaving vehicle through its simulated sensors and estimates Time To Collide (TTC) value. Each coalition member then executes the selected planning algorithm to find cooperative mitigation action plans that avoid the collisions for the planning horizon h. The TTC value in our experiments is set to 15 cycles. We choose the planning horizon h to 60 cycles, four times the TTC value. The reason is that the coalition members should not



(a) Acceleration misbehavior: Misbehaving vehicle accelerates from behind the coalition

(b) Full stop misbehavior: Misbehaving vehicle stops in front of the coalition

(c) Zigzag misbehavior: Misbehaving vehicle accelerates in a zigzag path from behind the coalition

Figure 5.1: Different types of misbehavior by the misbehaving vehicle

only avoid the detected collision at TTC=15 but avoid possible collisions even after TTC until the coalition is out of danger with respect to the misbehaving vehicle.

For our algorithm and DeCoC-MCTS, vehicle actions have fixed duration Δt. We set this action duration to 10 cycles for realistic action execution in MATISSE. Since the planning horizon is 60 and the action duration is 10, each CAV deliberates to select the next 6 actions. The IP centralized algorithm by Nakamura et al. generates waypoints at each timestep that represent the vehicle trajectory. The duration between two timesteps is fixed in this algorithm. We set the timestep duration to 3 cycles. For the planning horizon of 60 cycles and timestep duration of 3 cycles, a total of 20 waypoints are computed by the IP algorithm. In our experiments, we consider three ways the misbehaving vehicle can collide with the coalition CAVs. The three different misbehavior types are listed below:

- Acceleration misbehavior- The misbehaving vehicle is positioned behind the coalition of CAVs in the middle lane and speeds at 5 units/cycle in a straight path.
- Full stop misbehavior- The misbehaving vehicle is positioned in front of the coalition of CAVs in the middle lane and comes to a full stop at 0 units/cycle.

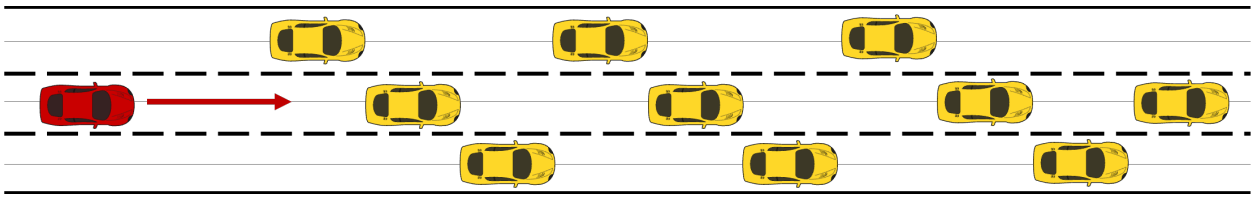


Figure 5.2: Scalability experiments: arrangement of CAVs in the coalition

- Zigzag misbehavior- The misbehaving vehicle is positioned behind the coalition of CAVs in the middle lane and speeds at 5 units/cycle traveling on a zigzag path between the first and middle lanes.

We perform simulation experiments for each misbehavior type to tune design parameters, test the algorithm's reliability, and test the algorithm's scalability. In the parameter tuning experiments and the reliability experiments, we consider a dense coalition of six CAVs as seen in Figure 5.1. In the scalability experiments, we consider varying numbers of CAVs in the coalition positioned in a zigzag manner as shown in Figure 5.2.

5.2 Tuning COCOA Parameters

We tune values for the design parameters C_M (see Equation 3.1), C_A (see Equation 3.3), and β (see Equation 3.5)using simulation experiments. Parameter tuning is carried out in an incremental manner, where we first tune the parameter C_M. After tuning C_M, we fix C_M to its ideal value and tune the parameter C_A. After tuning C_A, we fix both C_M and C_A to their ideal values and tune the parameter β. Moreover, we argue that the ideal parameter values can vary for different misbehavior types and confirm this using the experimental results.

For C_M parameter tuning, we employ the branching factor reduction method with the random action selection policy (Patel and Zalila-Wenkstern, 020a) instead of the intelligent action selection policy. The coalition-state-based reward weights update is also not performed to avoid setting the β value. For each value of C_M from [1,10,100,1000,10000] and for each misbehavior type from Figure 5.1, we perform 10 simulation experiments and note the

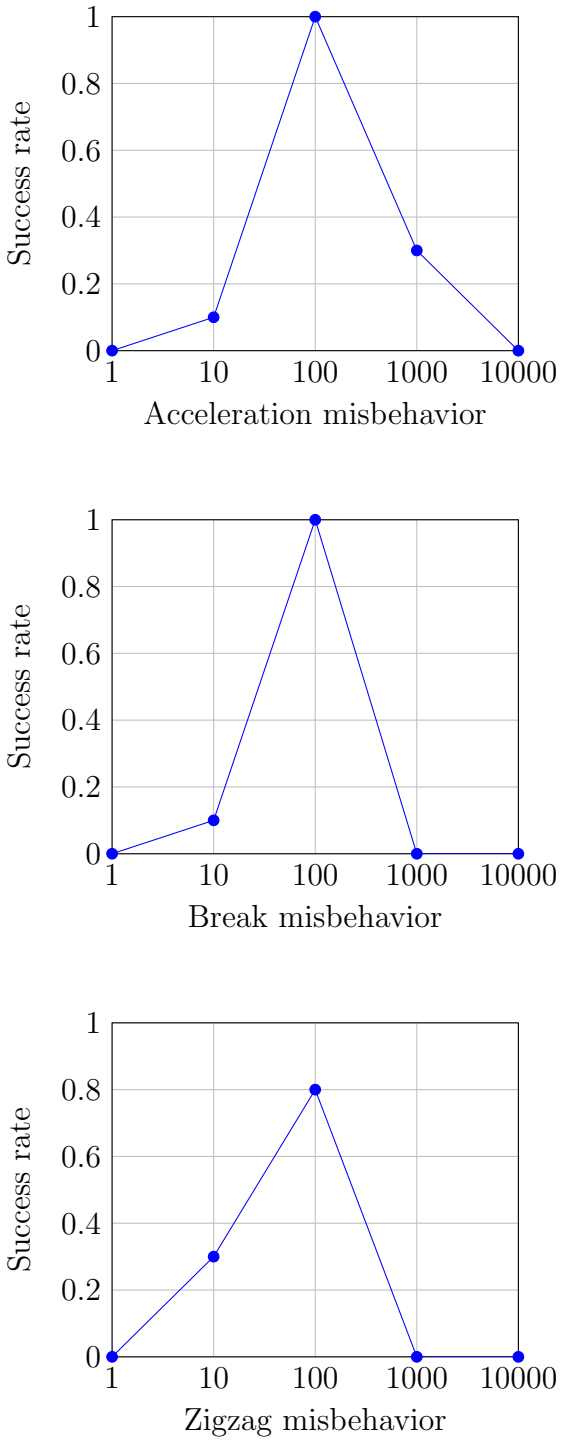


Figure 5.3: Tuning C_M parameter values

collision avoidance success rate. We plot the collision avoidance success rate versus C_M values (see Figure 5.3). Experimental results show that the C_M value of 100 works best for all misbehavior types.

For C_A parameter tuning, we employ the branching factor reduction method with the smart action selection policy using the C_A parameter. We fix the C_M value to 100 for all misbehavior types experiments. The coalition-state-based reward weights update is not performed to avoid setting the β value. We choose C_A values from [1,10,100,1000,10000]. We perform 10 simulation experiments for each C_A value and for each misbehavior type. Figure 5.4 shows the collision avoidance success rate versus C_A values for different misbehavior types. Experimental results show that the C_A value of 100 works best for the acceleration misbehavior and the zigzag misbehavior. However, for the break misbehavior, the C_A value of 1000 works better.

For β parameter tuning, we finally employ the adaptive reward weights update technique. We set the C_M value to 100 for all three misbehavior types. For the C_A parameter, we set 100 for the acceleration and zigzag misbehavior types and 1000 for the break misbehavior type. β values are chosen from [0,0.2,0.4,0.6,0.8,1]. For the acceleration misbehavior, we perform 10 simulation experiments for each β value. For the break and zigzag misbehavior types, we had to perform 20 simulation experiments for each β value in order to reduce the sampling error and get more accurate results. We plot the collision avoidance success rate versus the β values in Figure 5.5.

We list optimal parameter values for each misbehavior type in Table 5.1.

5.3 Reliability Evaluation

We define reliability as a collision avoidance algorithm's ability to find a non-colliding action plan for the coalition of CAVs in the given processing time. We consider a coalition of six CAVs and three different misbehavior types as shown in Figure 5.1. For each misbehavior

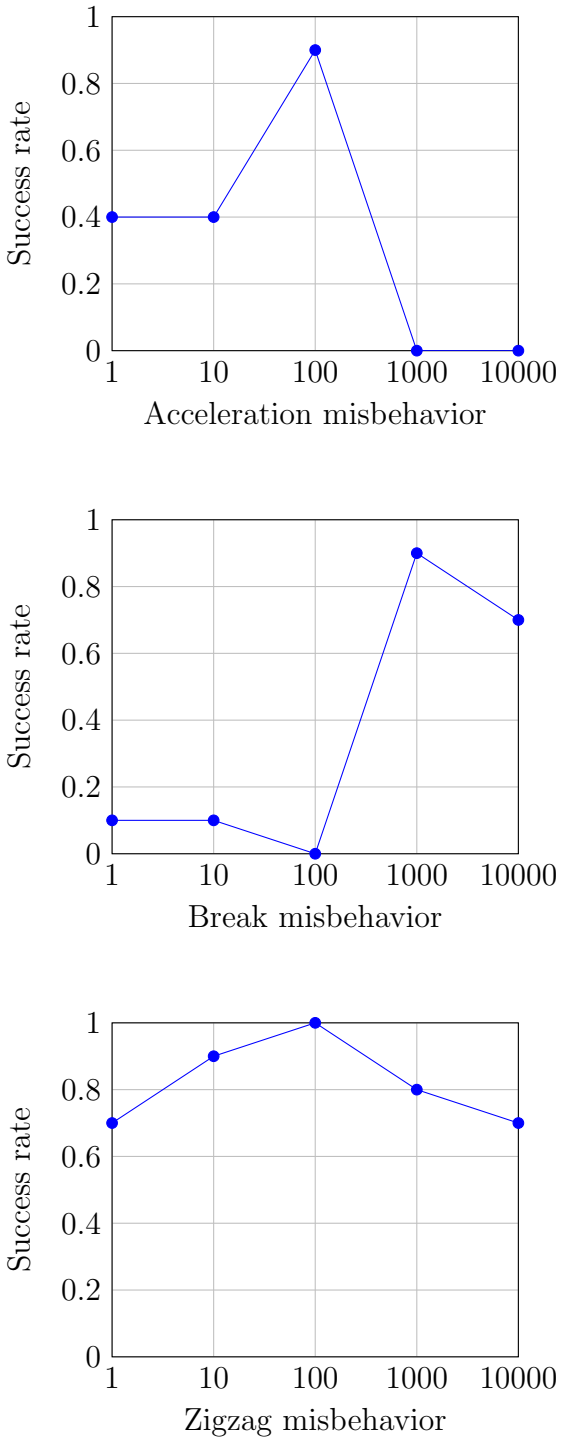


Figure 5.4: Tuning C_A parameter values

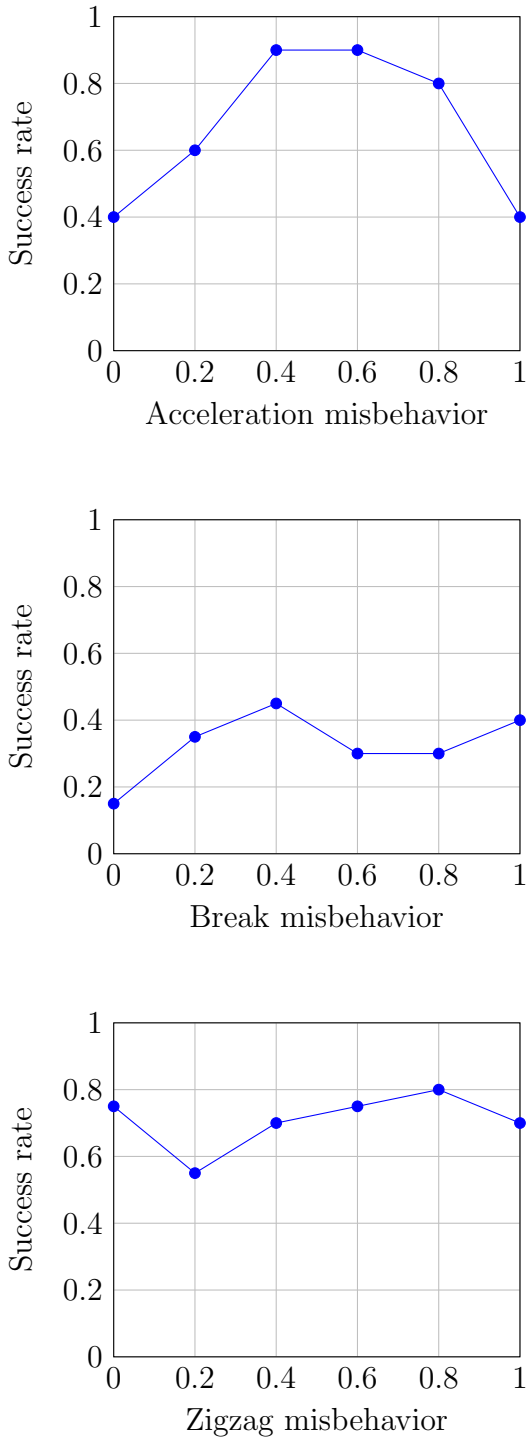
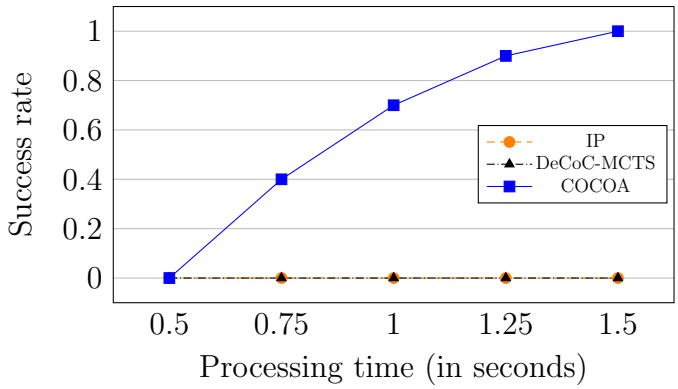
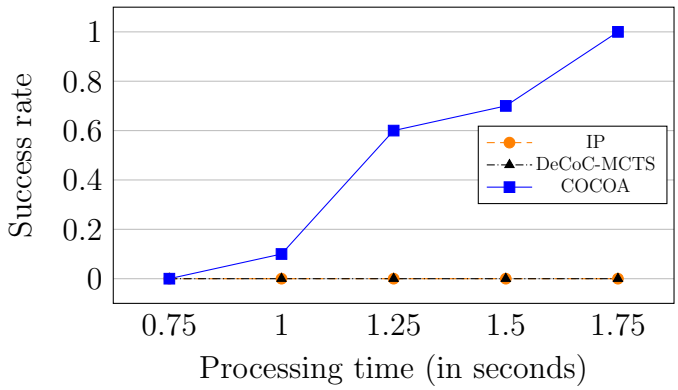


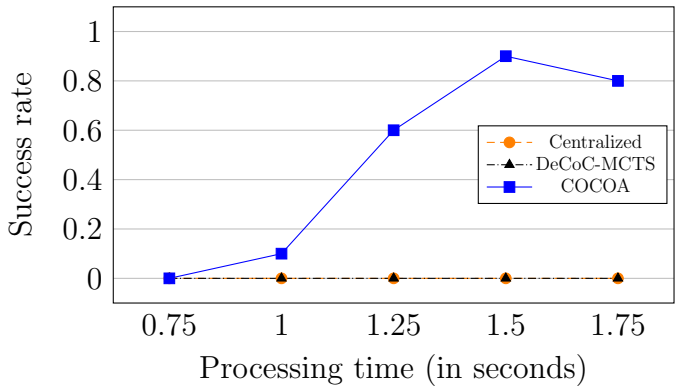
Figure 5.5: β parameter values



(a) Acceleration misbehavior



(b) Break misbehavior



(c) Zigzag misbehavior

Figure 5.6: Reliability results

Table 5.1: Optimal parameter values

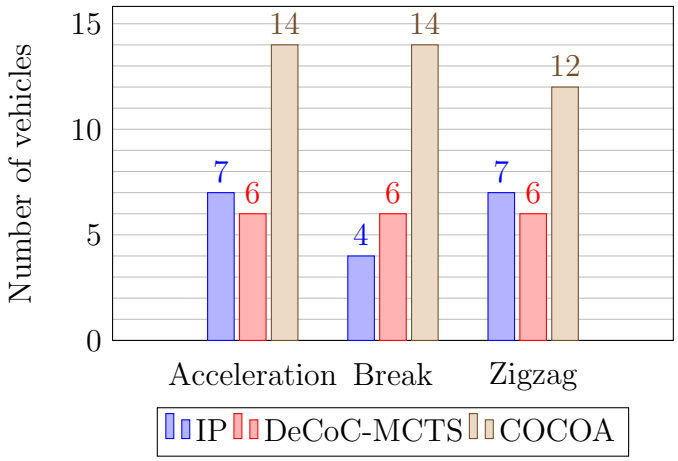
Misbehavior type	C_M	C_A	β
Acceleration	100	100	0.4
Break	100	1000	0.4
Zigzag	100	100	0.8

type, we set the processing time to a fixed value from a set of five values and find the collision avoidance success rate for 10 simulation experiments for each processing time value. We plot the collision avoidance success rate versus the processing time for COCOA, DeCoC-MCTS, and the centralized IP algorithms in Figure 5.6. DeCoC-MCTS and IP algorithms do not find successful action plans up to six vehicles, so they have zero success rates for all processing time values. The success rate for COCOA consistently improves as the processing time increases. For zigzag behavior, we can see a dip in the success rate for the processing time value of 1.75s than 1.5s. This is due to the sampling error caused by an insufficient number of samples. As the number of simulation experiments grows higher, we expect such errors to disappear.

5.4 Scalability Evaluation

Scalability is defined as the maximum number of CAVs for which the algorithm can complete its execution in a reasonable amount of time. For scalability experiments, we use the CAVs coalition structure as shown in Figure 5.2 and fix the maximum allowed processing time to 10 seconds. Figure 5.7 shows that the IP centralized algorithm can find the solution for up to 7 CAVs for the acceleration misbehavior type, 4 CAVs for the break misbehavior, and 7 CAVs for the zigzag misbehavior. The DeCoC-MCTS algorithm is able to complete its execution for up to 6 CAVs for all misbehavior types under 10 seconds but is not able to find a successful collision avoiding solution for any of these misbehavior types. We can see

Figure 5.7: Scalability results



that COCOA can find a successful collision-avoiding solutions for up to 14 CAVs for the acceleration and break misbehavior types and for up to 12 CAVs for the zigzag misbehavior type. COCOA considerably improves upon the scalability limits of the IP centralized and DeCoC-MCTS algorithms.

CHAPTER 6

COCOA ALGORITHMS FOR MULTIPLE COALITIONS

6.1 Model Definition

We assume that a set of n CAVs are navigating on a straight highway. These CAVs are distributed in a sequence of coalitions denoted by $\mathcal{N}$. Each coalition is defined by its ID $c \in \mathcal{N}$ and consists of n^c number of CAVs as its members. In each coalition c, one coalition member l^c is assigned the role of a leader and is responsible for coalition management. A single CAV's model including its state, actions, and mitigation action plan remains the same as described in Chapter 3. We define the coalition mitigation action plan α_h^c as a set of individual CAV mitigation action plans, one for each coalition member i, i.e, $\alpha_h^c = \{\alpha_h^i | i \in \mathcal{C}\}$.

We require that all coalitions in set $\mathcal{N}$ are navigating in a non-overlapping sequence on the highway, meaning that no two CAVs from two different coalitions are navigating laterally together in different lanes. This assumption allows us to define the definitions of preceding and succeeding coalitions. A *preceding coalition* or a *predecessor* of the coalition c is the coalition that is navigating ahead of the coalition c and is denoted by c_{pred}. A *succeeding coalition* or a *successor* of the coalition c is the coalition that is navigating after the coalition c and is denoted by c_{succ}. The coalition affected by the misbehaving vehicle is called *primary coalition* in the decision-making context. Coalitions other than the primary coalition are called *secondary coalitions*. We assume that the primary coalition is always located at either the beginning or at the end of the coalition sequence.

6.2 General Approach

Our general approach for the multi-coalition collision avoidance decision-making works in a sequential manner starting from the primary coalition along the coalition sequence $\mathcal{N}$. In the sequential decision-making approach, the task of choosing the final coalition action plan

for any coalition is performed by the next neighboring coalition in the coalition sequence $\mathcal{N}$. Our sequential decision-making approach uses two main algorithms: a primary coalition algorithm, and a secondary coalition algorithm.

We now describe the sequential decision-making approach. When a CAV from the primary coalition detects the misbehaving vehicle, the coalition leader of the primary coalition uses the primary coalition algorithm to select the top three coalition action plans according to Q^* values (See Chapter 3). Since the primary coalition is located either at the beginning or at the end of the coalition sequence $\mathcal{N}$, it only has one neighboring secondary coalition. The primary coalition sends the three action plans to the coalition leader of its neighboring secondary coalition. The secondary coalition leader uses the secondary coalition algorithm to choose one of the three action plans as the final plan for the primary coalition and sends it back to the primary coalition leader. The secondary coalition algorithm also generates three coalition action plans for the secondary coalition itself, which are then forwarded to the next secondary coalition leader for the final plan selection. This process is repeated until we reach the last secondary coalition in the coalition sequence $\mathcal{N}$. The last secondary coalition uses the secondary coalition algorithm to generate a single best coalition action plan for itself and sends it to its members.

6.3 Algorithms

We have two main algorithms used in the multiple coalitions collision avoidance problem: a primary coalition algorithm, and a secondary coalition algorithm.

6.3.1 Primary coalition algorithm

We now describe the primary coalition algorithm, given in Algorithm 8. Without any loss of generality, assume that the primary coalition c is the last coalition in the sequence $\mathcal{N}$. When any CAV of the primary coalition c is affected by the misbehaving vehicle m, it sends

Algorithm 8 Primary coalition algorithm

Require: $T_i, \forall i \in \mathcal{C}$
Ensure: $\alpha_h^i, \forall i \in \mathcal{C}$
1: $currentNodes \leftarrow$ null
2: **for** i in $\mathcal{C}$ **do**
3: $\quad currentNodes(i) \leftarrow T_i.n_\mu$
4: **end for**
5: $\{\alpha_h^c\} \leftarrow$ RecursivePlansSelection($currentNodes$)
6: $predecessor \leftarrow$ getPredecessorCoalition()
7: $successor \leftarrow$ getSuccessorCoalition()
8: **if** $predecessor \neq$ null **then**
9: $\quad$ sendPlans($predecessor, \{\alpha_h^c\}$)
10: $\quad \alpha_h^c \leftarrow$ receiveFinalPlan($predecessor$)
11: **else**
12: $\quad$ **if** $successor \neq$ null **then**
13: $\quad\quad$ sendPlans($successor, \{\alpha_h^c\}$)
14: $\quad\quad \alpha_h^c \leftarrow$ receiveFinalPlan($successor$)
15: $\quad$ **else**
16: $\quad\quad \alpha_h^c \leftarrow \alpha_h^c[1]$
17: $\quad$ **end if**
18: **end if**
19: **for** i in $\mathcal{C}$ **do**
20: $\quad \alpha_h^i \leftarrow \alpha_h^c(i)$
21: **end for**

an alert to all coalition members of c. Upon receiving the alert, each coalition member i of c executes the COCOA algorithms for a single coalition described in Chapter 3 to generate an MCTS tree of CAV states and CAV actions and to estimate Q^* values for each CAV action in the MCTS tree. Each CAV shares its MCTS tree with estimated Q^* values with the coalition leader l^c. Each path from the tree's root node to the leaf node represents an individual mitigation action plan prioritized by Q^* values. The coalition leader receives the individual mitigation action plans in form of MCTS trees from all members and selects a set of top three non-colliding coalition mitigation action plans α_h^c ranked using Q^* values in step 5 of the Algorithm 8. It uses RecursivePlanSelection() function in order to derive top three mitigation action plan, which is quite similar to the RecursiveCombinationSelection() function in the Algorithm 7 from Chapter 3. The primary coalition leader l^c sends the three

coalition action plans to its preceding coalition c''s leader $l^{c'}$ in step 9. The task of choosing the final coalition action plan for the primary coalition c is left to $l^{c'}$, since the final plan can affect the CAVs of the coalition c'. When l^c receives the final action plan α_h^c from $l^{c'}$ in step 10, it extracts individual plans from this final plan in steps 19-21 and sends it to its members for execution.

6.3.2 Secondary coalition algorithm

We now describe the algorithm for secondary coalitions, given in Algorithm 9. Assume that a leader CAV $l^{c'}$ of the secondary coalition c' received a set of three coalition action plans from its succeeding coalition leader l^c. Upon receiving the set of coalition action plans, $l^{c'}$ determines the impact for all c's action plans (step 2-4 in Algorithm 9) and chooses the one which has the least impact on the members of c' (step 5). $l^{c'}$ sends the chosen plan α_h^c back to l^c as the final coalition action plan for c. $l^{c'}$ also sends the message to all its members to derive their individual mitigation action plans (steps 6-9). This message includes c's plan α_h^c too as it affects the members of c'. Upon receiving this message, each CAV i of the coalition c' executes COCOA algorithms for a single coalition to derive its set of individual mitigation action plans α_h^i such that they avoid collisions with the coalition c's plan α_h^c. Each member sends its set of individual mitigation action plans to the leader $l^{c'}$. Upon receiving the individual action plans from all members, leader $l^{c'}$ selects a set of top three non-colliding coalition mitigation action plans $\alpha_h^{c'}$ using RecursivePlanSelection() function (step 14 in Algorithm 9), same as the one used in the primary coalition algorithm. Since primary coalition c was the successor of c', it checks if there is any preceding coalition. If there is a preceding coalition to c', $l^{c'}$ sends the three plans to its preceding coalition's leader (in steps 26-27), which is responsible for choosing the final plan for c'. If there is no preceding coalition left, $l^{c'}$ chooses the best coalition action plan $\alpha_h^{c'}$ according to Q^* values (in step 29). If c were the predecessor of c', $l^{c'}$ would have sent the three plans to its succeeding

Algorithm 9 Secondary coalition c' algorithm

Require: $\{\alpha_h^c\}$
Ensure: $\alpha_h^i, \forall i \in \mathcal{C}'$
1: Initialize $impactFactors$ to a null map
2: **for** i in 1,2,3 **do**
3: $\quad impactFactors(i) \leftarrow$ computeImpactScore($\{\alpha_h^c\}[i]$)
4: **end for**
5: $\alpha_h^c \leftarrow$ pickMinImpactPlan($\{\alpha_h^c\}, impactFactors$)
6: sendToCoalition(c, α_h^c)
7: **for** i in $\mathcal{C}'$ **do**
8: $\quad T_i \leftarrow$ sendRequestForIndividualPlans(i, α_h^c)
9: **end for**
10: $currentNodes \leftarrow$ null
11: **for** i in $\mathcal{C}$ **do**
12: $\quad currentNodes(i) \leftarrow T_i.n_\mu$
13: **end for**
14: $\{\alpha_h^c\} \leftarrow$ RecursivePlansSelection($currentNodes$)
15: $predecessor \leftarrow$ getPredecessorCoalition()
16: $successor \leftarrow$ getSuccessorCoalition()
17: **if** $c = predecessor$ **then**
18: $\quad$ **if** $successor \neq$ null **then**
19: $\quad\quad$ sendPlans($successor, \{\alpha_h^c\}$)
20: $\quad\quad \alpha_h^c \leftarrow$ receiveFinalPlan($successor$)
21: $\quad$ **else**
22: $\quad\quad \alpha_h^c \leftarrow \alpha_h^c[1]$
23: $\quad$ **end if**
24: **else**
25: $\quad$ **if** $predecessor \neq$ null **then**
26: $\quad\quad$ sendPlans($predecessor, \{\alpha_h^c\}$)
27: $\quad\quad \alpha_h^c \leftarrow$ receiveFinalPlan($predecessor$)
28: $\quad$ **else**
29: $\quad\quad \alpha_h^c \leftarrow \alpha_h^c[1]$
30: $\quad$ **end if**
31: **end if**
32: **for** i in $\mathcal{C}'$ **do**
33: $\quad \alpha_h^i \leftarrow \alpha_h^c(i)$
34: **end for**

coalition leader (in steps 19-20) if one existed. After receiving the final action plan from the preceding coalition leader or deciding it by itself, $l^{c'}$ extracts individual this final plan to its members for execution in steps 32-34.

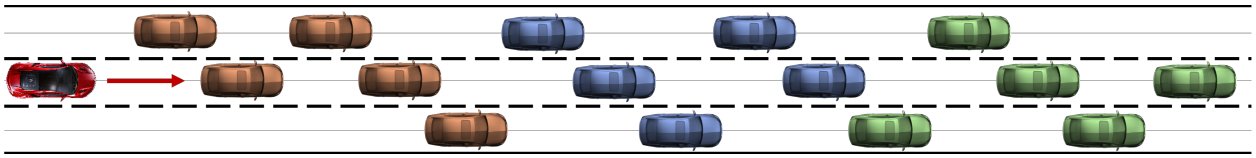
CHAPTER 7

MULTIPLE COALITIONS ALGORITHMS EVALUATION USING SIMULATION EXPERIMENTS

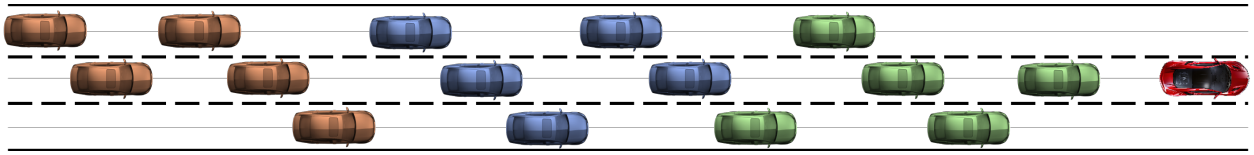
In this chapter, we evaluate COCOA algorithms for multi-coalition collision avoidance by implementing them in MATISSE simulator (Torabi et al., 2018). We obtain data from simulation experiments to choose values of COCOA design parameters for different misbehavior types. We use the simulation experimental data to determine the reliability and the scalability of the multi-coalition approach. Additionally, we perform a trade-off analysis between the individual coalition size and the number of coalitions on the road for a fixed number of CAVs.

7.1 Simulation Experimental Setting

In the virtual Matisse simulation environment, the unit of time is called *cycle*, and the unit of length is simply called *unit*. In our multi-coalition simulation experiments, a sequence of coalitions is navigating on a three-lane highway. Each CAV member in each coalition of the coalition sequence is cruising at the speed of 2 units/cycle. During the simulation experiment, we add a misbehaving virtual CAV to the simulation environment that misbehaves to cause a potential collision with one or more coalition CAVs. When an affected CAV detects the misbehaving vehicle through one of its simulated sensors, it sends an alert to all of its coalition members. The coalition containing the affected CAV is known as the primary horizon. In our experiments, the primary coalition is located at either end of the coalition sequence. In our experiments, we use the planning horizon $h = 60$ cycles, the Time To Collide (TTC) value $TTC = 20$ cycles, and the action duration $\delta t = 10$ cycles. Each coalition deliberates to choose a coalition action plan containing 6 consecutive actions for each of its coalition members. We perform three types of experimental evaluation: reliability evaluation, scalability evaluation, and trade-off analysis.



(a) Acceleration misbehavior: Misbehaving vehicle accelerates from behind three coalitions



(b) Full stop misbehavior: Misbehaving vehicle breaks in front of three coalitions

Figure 7.1: Misbehavior types for multi-coalition reliability evaluation

In the parameter tuning experiments and the reliability evaluation experiments, we consider three coalitions in the coalition sequence. Each coalition consists of 5 coalition members. We consider two types of misbehaviors by the misbehaving vehicle (see Figure 7.1) as listed below:

- Acceleration misbehavior- The misbehaving vehicle is positioned behind the primary coalition in the middle lane and speeds at 5 units/cycle in a straight path.
- Full stop misbehavior- The misbehaving vehicle is positioned in front of the primary coalition in the middle lane and comes to a full stop at 0 units/cycle.

7.2 Parameter Tuning

We carried out parameter tuning experiments to tune single coalition COCOA algorithm parameters for our multi-coalition scenarios in Figure 7.1, namely: C_M (see Equation 3.1), C_A (see Equation 3.3), and β (see Equation 3.5). For parameters C_M and C_A, we did not find difference in the tuned parameter values as given in Section 5.2. We found different tuned values for the β parameter for the break misbehavior type. For the acceleration misbehavior, we found $\beta = 0.6$ to be the optimum value. This is not in contrast to the single coalition experiments, as 0.6 is also one of the optimum values for single coalition experiments along

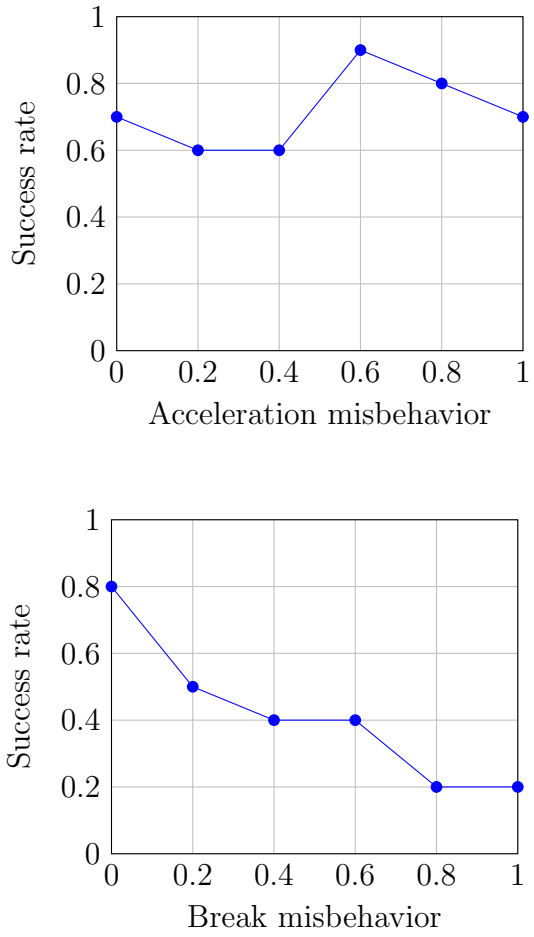


Figure 7.2: β parameter values for multi-coalition experiments

with 0.4 (see Figure 7.2). However, for the break misbehavior, we found $\beta = 0$ to be the optimum value. This is a rare scenario where the adaptive reward function design does not improve upon the standard reward function design.

7.3 Reliability Evaluation

Here, we define reliability as a collision avoidance algorithm's ability to find a non-colliding coalition action plan for each of the three coalitions in the given computational processing time. We consider three coalitions, each consisting of five CAVs, and two misbehavior types

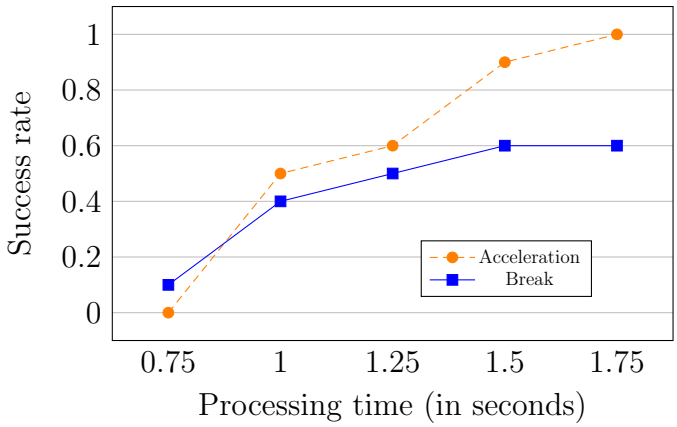


Figure 7.3: Reliability evaluation

as shown in Figure 7.1. For each misbehavior type, we set the processing time to a fixed value from a set of five values and find the collision avoidance success rate for 10 simulation experiments for each processing time value. We plot the collision avoidance success rate versus the processing time for both misbehavior types in Figure 7.3. We can see that the success rate increases as the allowed computation time increases.

7.4 Scalability Evaluation

We define scalability for multi-coalition decision making as the maximum number of coalitions for which the algorithm can complete its execution in a reasonable amount of time. In the scalability experiments, we fix the number of CAVs in each coalition to 5 and vary number of coalitions on the road from one to up to five coalitions. We use the acceleration misbehavior type to test the scalability of our multi-coalition approach. We fix the allowed processing time to 10 seconds in order to evaluate the computational efficiency of our algorithm. Since the multi-coalition decision-making algorithm is sequential in nature, each coalition can start its execution of the decision-making algorithm only after the previous coalition has finished. For this reason, each coalition gets the allowed computation time in seconds that equals to 10 divided by the number of coalitions $|\mathcal{N}|$ in the coalition sequence.

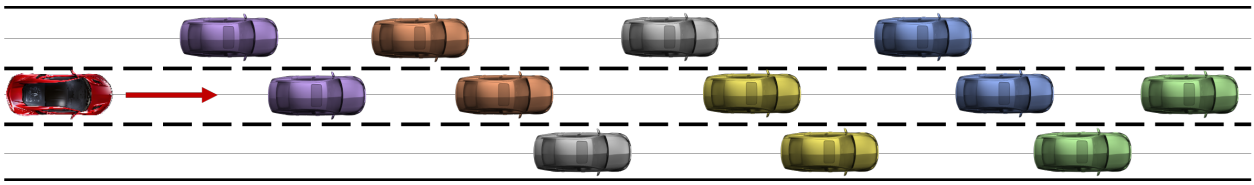


Figure 7.4: Trade-off analysis: 6 coalitions of 2 CAVs each (total 12 CAVs)

All CAVs in a given coalition perform decision-making in parallel and independent of each other for full $10/|\mathcal{N}|$ seconds. Experimental results show that COCOA multi-coalition algorithm found non-colliding coalition action plans for up to 4 coalitions in the coalition sequence, where each coalition consisted of five CAVs. It means that a total number of 20 CAVs can perform collision avoidance decision-making in 10 seconds of processing time, which is a significant improvement over the 14 CAVs limit of the single coalition COCOA algorithms (see Figure 5.7).

7.5 Trade-off Analysis

We perform the trade-off analysis between the number of coalitions and the individual coalition size for a fixed total number of CAVs on the road. For a fixed total number of CAVs, we can reduce each individual coalition size to increase the number of coalitions and vice versa. Our objective is to derive an argument that for the best collision avoidance results, one needs to balance between the individual coalition size and the number of coalitions. We fix the total number of CAVs to 12 (See Figure 7.4). We generate several different coalitions structures including 12 coalitions of single CAV each, 6 coalitions of 2 CAVs each, 4 coalitions of 3 CAVs each, 3 coalitions of 4 CAVs each, 2 coalitions of 6 CAVs each, and a single coalition of 12 CAVs. For each of these coalitions structures, we perform 10 simulations experiments in MATISSE and note the collision avoidance success rate. We fix the allowed computation time to 3.6 seconds. For the sequential decision-making algorithm used for the multi-coalitions, 3.6 seconds computation time gets divided among the coalitions. For 12

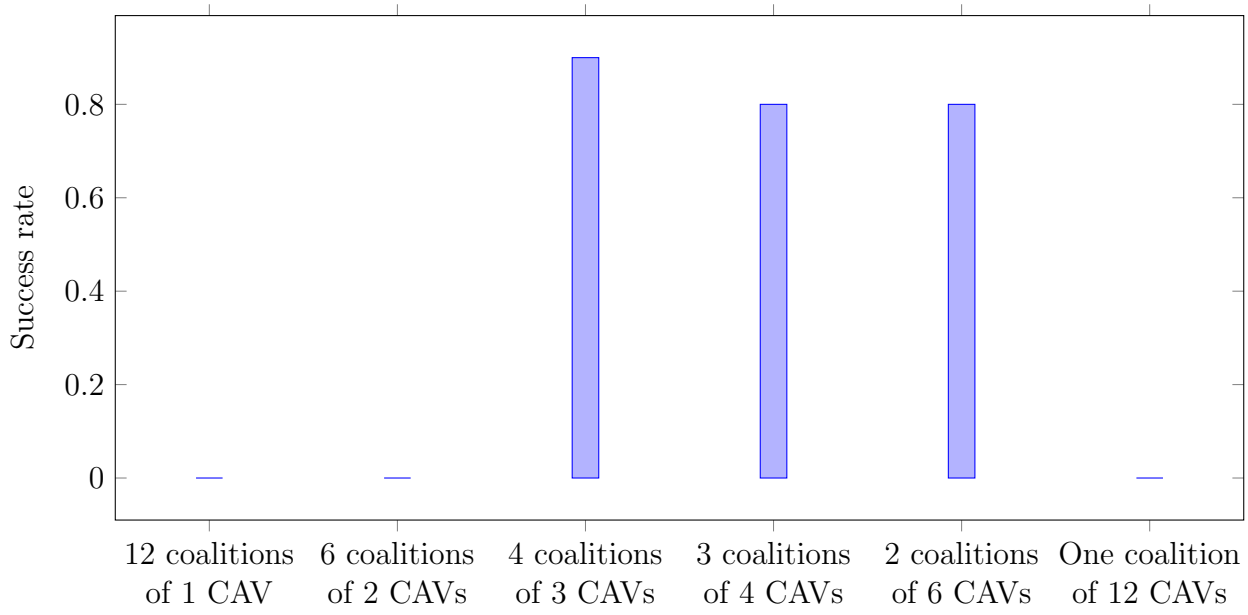


Figure 7.5: Success rate for different number of coalitions constructed out of 12 CAVs

coalitions, each coalition gets 0.3 seconds of the computation time. Whereas for 2 coalitions, each coalition gets 1.8 seconds of the computation time.

We can see that 4 coalitions with 3 CAVs each configuration yields the highest success rate (see Figure 7.5). 3 coalitions with 4 CAVs each and 2 coalitions with 6 CAVs each configurations also have very good success rate values. However, very small coalition sizes of 1 and 2, as well as very large coalition size of 12, do not yield good results at all. This confirms our initial argument that the balance between the number of coalitions and the individual coalition size must be struck to yield the best results for collision avoidance.

CHAPTER 8

CONCLUSION

In this work, we presented COCOA, a set of **co**operative **co**llision **a**voidance algorithms for coalitions of autonomous vehicles in the presence of a misbehaving vehicle.

8.1 Contributions

Among the unique contributions of our work, we mention the following:

1. We used the concept of CAV coalition for collision avoidance task, where coalition members continuously exchange information with each other.

2. Unlike existing works which focus on either centralized or decentralized solutions, we proposed a partially-decentralized hierarchical decision-making approach that is achieved at two decision-making levels, i.e., individual vehicle-level and coalition leader-level.

3. We proposed the first cooperative collision avoidance algorithm for multiple coalitions of CAVs to our best knowledge.

4. The proposed COCOA algorithm improved upon the state-of-the-art MCTS-based CAV algorithms by exponentially reducing the size of the search tree. With COCOA, it is computationally feasible to find solutions for a larger number of agents (Patel and Zalila-Wenkstern, 020a, 021a).

5. The proposed COCOA algorithm introduces a novel Intelligent Action Selection policy for MCTS-based algorithms that employ a reduced branching factor (Patel and Zalila-Wenkstern, 021a).

6. We proposed a novel Adaptive Reward Function design that improves the reliability of MCTS (Patel and Zalila-Wenkstern, 021b).

7. We developed and implemented a CAV model with radar and LIDAR sensors in MATISSE 3.0, a multi-agent-based simulation software to enable realistic evaluation and visualization of COCOA algorithms.

8. We conducted extensive experiments in the multi-agent-based simulator to compare the reliability and the scalability of COCOA algorithms with the leading centralized algorithm proposed by Nakamura et al. (2020) and the decentralized AI-based algorithm proposed by Kurzer et al. (018b).

9. We justified the need for multiple coalitions structure for efficient collision avoidance through experimental analysis.

8.2 Lessons learned

Lessons learned during this research include:

1. Decentralized reinforcement learning methods are not suitable as-is for critical applications such as the collision avoidance of autonomous vehicles, since the individual decisions by agents are not synchronized to check for conflicts.

2. Vehicle-to-vehicle communication is essential in achieving explicit cooperation among autonomous vehicles, therefore its usage should be encouraged.

3. Monte Carlo Tree Search although suitable for very large state-space problems can not handle a large number of agents in multi-agent problems.

4. Vehicle coalition structure generation is a crucial component for developing a successful cooperative action planning system for autonomous vehicles.

5. For a fixed number of vehicles on the road, a balance between the number of coalitions formed and the individual coalition size helps in achieving a better collision avoidance rate.

8.3 Future Work

Future work includes the investigation of the following problems:

1. COCOA algorithms perform collision avoidance for multiple coalitions up to the planning horizon. We plan to extend it to perform collision avoidance past the planning horizon by starting a new COCOA decision-making cycle.

2. Current adaptive reward function yields better results than the standard reward function in most scenarios but some. We plan to analyze it thoroughly to be able to determine the usefulness of the adaptive reward weights for any given problem.

3. In COCOA algorithms, we use a finite and discrete set of individual CAV actions for the collision avoidance task. It would be interesting to investigate ways to include a continuous set of individual CAV actions while retaining the scalability advantage of COCOA algorithms.

4. Current COCOA algorithm for multiple coalitions requires coalitions to be in a non-overlapping sequence. It would be interesting to extend it to allow multiple coalitions to navigate laterally together.

REFERENCES

Abboud, K., H. A. Omar, and W. Zhuang (2016). Interworking of dsrc and cellular network technologies for v2x communications: A survey. *IEEE transactions on vehicular technology 65*(12), 9457–9470.

Ahmad, I., R. M. Noor, I. Ali, M. Imran, and A. Vasilakos (2017). Characterizing the role of vehicular cloud computing in road traffic management. *International Journal of Distributed Sensor Networks 13*(5), 1550147717708728.

Andriotis, C. and K. Papakonstantinou (2019). Managing engineering systems with large state and action spaces through deep reinforcement learning. *Reliability Engineering & System Safety 191*, 106483.

Audi. Audi gives up on level 3 autonomous driver-assist system in a8. https://www.motorauthority.com/news/1127984_audi-gives-up-on-level-3-autonomous-driver-assist-system-in-a8. Accessed on 2020-12-02.

beamsearch. Beam search. https://en.wikipedia.org/wiki/Beam_search. Accessed on 2021-06-10.

Bellan, D. and C. Wartnaby (2020). Decentralized cooperative collision avoidance for automated vehicles: a real-world implementation. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 487–494. IEEE.

Bhatia, A., K. Haribabu, K. Gupta, and A. Sahu (2018). Realization of flexible and scalable vanets through sdn and virtualization. In *2018 International conference on information networking (ICOIN)*, pp. 280–282. IEEE.

BMW. BMW Takes Self-Driving to Level 3 Automation. https://www.electronicdesign.com/markets/automotive/article/21136427/bmw-takes-selfdriving-to-level-3-automation. Accessed on 2020-12-02.

Boutilier, C. (1999). Sequential optimality and coordination in multiagent systems. In *IJCAI*, Volume 99, pp. 478–485.

Boyd, S., N. Parikh, and E. Chu (2011). *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc.

Browne, C. B., E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games 4*(1), 1–43.

Burger, C. and M. Lauer (2018). Cooperative multiple vehicle trajectory planning using miqp. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 602–607. IEEE.

Committee, S. A. V. S. (2016). Sae j3016: Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems. *SAE J3016, Sep*.

Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on Computers and Games*, pp. 72–83. Springer.

Djukic, P. and S. Valaee (2018). 802.16 mesh networking. In *WiMAX*, pp. 161–188. CRC Press.

Duering, M., K. Franke, R. Balaghiasefi, M. Gonter, M. Belkner, and K. Lemmer (2014). Adaptive cooperative maneuver planning algorithm for conflict resolution in diverse traffic situations. In *2014 International Conference on Connected Vehicles and Expo (ICCVE)*, pp. 242–249. IEEE.

Duering, M. and P. Pascheka (2014). Cooperative decentralized decision making for conflict resolution among autonomous agents. In *2014 IEEE International Symposium on Innovations in Intelligent Systems and Applications (INISTA) Proceedings*, pp. 154–161. IEEE.

Elliott, D., W. Keen, and L. Miao (2019). Recent advances in connected and automated vehicles. *Journal of Traffic and Transportation Engineering (English Edition) 6*(2), 109–131.

Ernst, T., V. Nebehaj, and R. Søråsen (2009). Cvis: Calm proof of concept preliminary results. In *2009 9th International Conference on Intelligent Transport Systems Telecommunications,(ITST)*, pp. 80–85. IEEE.

Gao, L., D. Chu, Y. Cao, L. Lu, and C. Wu (2019). Multi-lane convoy control for autonomous vehicles based on distributed graph and potential field. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 2463–2469. IEEE.

Ghafoor, K. Z., M. Guizani, L. Kong, H. S. Maghdid, and K. F. Jasim (2019). Enabling efficient coexistence of dsrc and c-v2x in vehicular networks. *IEEE Wireless Communications 27*(2), 134–140.

GM. "Ultra Cruise" to take GM's Super Cruise hands-free driving system into the city. https://www.motorauthority.com/news/1128224_ultra-cruise-to-take-gm-s-super-cruise-hands-free-driving-system-into-the-city. Accessed on 2020-12-02.

Hawkins, A. J. (2020, January). LIDAR sensors aren't just for self-driving cars anymore. https://www.theverge.com/2020/1/7/21055011/lidar-sensor-self-driving-mainstream-mass-market-velodyne-ces-2020. Accessed on 2020-12-02.

Johnson, C. (2017). Readiness of the road network for connected and autonomous vehicles. *RAC Foundation: London, UK*.

Kessler, T. and A. Knoll (2019). Cooperative multi-vehicle behavior coordination for autonomous driving. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1953–1960. IEEE.

Krajewski, R., P. Themann, and L. Eckstein (2016). Decoupled cooperative trajectory optimization for connected highly automated vehicles at urban intersections. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pp. 741–746. IEEE.

Kurzer, K., F. Engelhorn, and J. M. Zöllner (2018b). Decentralized cooperative planning for automated vehicles with continuous monte carlo tree search. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 452–459. IEEE.

Kurzer, K., M. Fechner, and J. M. Zöllner (2020). Accelerating cooperative planning for automated vehicles with learned heuristics and monte carlo tree search. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1726–1733. IEEE.

Kurzer, K., C. Zhou, and J. M. Zöllner (2018a). Decentralized cooperative planning for automated vehicles with hierarchical monte carlo tree search. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 529–536. IEEE.

Kuutti, S., S. Fallah, K. Katsaros, M. Dianati, F. Mccullough, and A. Mouzakitis (2018). A survey of the state-of-the-art localization techniques and their potentials for autonomous vehicle applications. *IEEE Internet of Things Journal 5*(2), 829–846.

Lan, L.-C., M.-Y. Tsai, T.-R. Wu, I. Wu, C.-J. Hsieh, et al. (2020). Learning to stop: Dynamic simulation monte-carlo tree search. *arXiv preprint arXiv:2012.07910*.

Lenz, D., T. Kessler, and A. Knoll (2016). Tactical cooperative planning for autonomous highway driving using monte-carlo tree search. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pp. 447–453. IEEE.

Lucero, S. (2016). C-v2x offers a cellular alternative to ieee 802.11 p/dsrc. *IHS TECHNOLOGY Internet Everything 3*, 1–3.

Manoochehri, H. and R. Wenkstern (2017). Dynamic coalition structure generation for autonomous connected vehicles. In *2017 IEEE International Conference on Agents (ICA)*, pp. 21–26. IEEE.

Manzinger, S., M. Leibold, and M. Althoff (2017). Driving strategy selection for cooperative vehicles using maneuver templates. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 647–654. IEEE.

Mercedes. Mercedes Takes '21 S-Class to Next Level of Autonomous Driving. https://www.wardsauto.com/vehicles/mercedes-takes-21-s-class-next-level-autonomous-driving. Accessed on 2020-12-02.

Milakis, D. (2019). Long-term implications of automated vehicles: An introduction.

Naik, G., B. Choudhury, and J.-M. Park (2019). Ieee 802.11 bd & 5g nr v2x: Evolution of radio access technologies for v2x communications. *IEEE Access 7*, 70169–70184.

Nakamura, A., Y.-C. Liu, and B. Kim (2020). Short-term multi-vehicle trajectory planning for collision avoidance. *IEEE Transactions on Vehicular Technology*.

OR-Tools. Google OR-Tools software. https://developers.google.com/optimization. Accessed on 2020-12-03.

Pascheka, P. and M. Duering (2015). Advanced cooperative decentralized decision making using a cooperative reward system. In *2015 International Symposium on Innovations in Intelligent SysTems and Applications (INISTA)*.

Patel, D. and R. Zalila-Wenkstern (2020a). Collaborative collision avoidance for cavs in unpredictable scenarios. In *2020 IEEE 3rd Connected and Automated Vehicles Symposium (CAVS)*, pp. 1–6. IEEE.

Patel, D. and R. Zalila-Wenkstern (2020b). *16 Connected and Automated Vehicles: Study of Platooning*, pp. 263 – 284. Berlin, Boston: De Gruyter.

Patel, D. and R. Zalila-Wenkstern (2021a). Scalable monte carlo tree search for cavs action planning in colliding scenarios. In *2021 IEEE 32nd Intelligent Vehicles Symposium (IV21)*, pp. 1–8 MC–MPS.14. IEEE.

Patel, D. and R. Zalila-Wenkstern (2021b). Adaptive reward for cav action planning using monte carlo tree search. In *2021 IEEE 24th International Conference on Intelligent Transportation (ITSC)*, pp. 1–7 MoA7.8. IEEE.

Rasshofer, R. H. and K. Gresser (2005). Automotive radar and lidar systems for next generation driver assistance functions. *Advances in Radio Science 3*.

Schaeffer, M. S. N. S. J., N. Shafiei, et al. (2009). Comparing uct versus cfr in simultaneous games. In *IJCAI Workshop on General Game Playing*. Citeseer.

Schwarting, W., J. Alonso-Mora, and D. Rus (2018). Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems 1*(1), 187–210.

Schwarting, W. and P. Pascheka (2014). Recursive conflict resolution for cooperative motion planning in dynamic highway traffic. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pp. 1039–1044. IEEE.

Siegel, J. E., D. C. Erb, and S. E. Sarma (2018, Aug). A survey of the connected vehicle landscape—architectures, enabling technologies, applications, and development areas. *IEEE Transactions on Intelligent Transportation Systems 19*(8), 2391–2406.

Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. (2017). Mastering the game of go without human knowledge. *nature 550*(7676), 354–359.

Tak, M. J., M. Lanctot, and M. H. Winands (2014). Monte carlo tree search variants for simultaneous move games. In *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–8. IEEE.

Tesla. Elon Musk Says Tesla Is 'Very Close' To Level 5 Self-Driving Technology. https://insideevs.com/news/433141/elon-musk-tesla-level-5-autonomous-driving/. Accessed on 2020-12-02.

Torabi, B., M. Al-Zinati, and R. Z. Wenkstern (2018). Matisse 3.0: A large-scale multi-agent simulation system for intelligent transportation systems. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pp. 357–360. Springer.

Vodopivec, T., S. Samothrakis, and B. Ster (2017). On monte carlo tree search and reinforcement learning. *Journal of Artificial Intelligence Research 60*, 881–936.

Wang, Z., Y. Zheng, S. E. Li, K. You, and K. Li (2018). Parallel optimal control for cooperative automation of large-scale connected vehicles via admm. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1633–1639. IEEE.

Yuan, Y., R. Tasik, S. S. Adhatarao, Y. Yuan, Z. Liu, and X. Fu (2020). Race: Reinforced cooperative autonomous vehicle collision avoidance. *IEEE transactions on vehicular technology 69*(9), 9279–9291.

Zhang, F., H. Stähle, G. Chen, C. C. C. Simon, C. Buckl, and A. Knoll (2012). A sensor fusion approach for localization with cumulative error elimination. In *2012 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pp. 1–6. IEEE.

Zhao, J., B. Liang, and Q. Chen (2018). The key technology toward the self-driving car. *International Journal of Intelligent Unmanned Systems*.

BIOGRAPHICAL SKETCH

Dhruvkumar Patel was born near Ahmedabad, India in 1992. After completing his schoolwork at Vishwa Vidyalaya High School in Ahmedabad in 2009, Dhruvkumar entered Dhirubhai Ambani Institute of Information and Communication Technology where he earned a Bachelor of Technology with a major in Information and Communication Technology in 2013. Dhruvkumar worked for the ecommerce company, Infibeam, during 2013 as a software developer and for TIFAC in 2014 as a research associate. In January 2015, he moved to Dallas, TX and entered The University of Texas at Dallas. He was awarded a MS in Computer Science from The University of Texas at Dallas in 2017 and immediately began working towards a PhD degree in Computer Science. He married Janki Patel in November, 2017.

CURRICULUM VITAE

Dhruvkumar Patel

November 16, 2021

Educational History:

BTech, Information and Communication Technology, DA-IICT, 2013
MS, Computer Science, The University of Texas at Dallas, 2017
PhD, Computer Science, The University of Texas at Dallas, 2021(*expected*)

Employment History:

Teaching/Research Assistant, The University of Texas at Dallas, January 2016 – May 2021
Course Grader, The University of Texas at Dallas, January 2015 – December 2015
Research Associate, TIFAC, New Delhi, India July 2014 – December 2014
Software Developer, Infibeam, Ahmedabad, India July 2013 – November 2013
Research Assistant, IRLab, DA-IICT, India July 2013 – December 2013
Research Intern, Bell Labs, Bangalore, India January 2013 – June 2013

Publications:

Dhruvkumar Patel and Rym Z. Wenkstern. Cooperative action planning for CAVs in colliding scenarios using Monte Carlo Tree Search. Submitted to IEEE transactions on Intelligent Transportation Systems (T-ITS), *Under review*.
Dhruvkumar Patel and Rym Z. Wenkstern. Adaptive Reward for CAV Action Planning Using Monte Carlo Tree Search. In Proceedings of the 24th IEEE Intelligent Transportation Systems Conference, ITSC 2021, page 1-7, (Virtual) Indianapolis, USA, September 2021.
Dhruvkumar Patel and Rym Z. Wenkstern. Scalable Monte Carlo Tree Search for CAVs Action Planning in Colliding Scenarios. In proceedings of the 32nd IEEE Intelligent Vehicles Symposium, IV 2021, page 1-8, (Virtual) Nagoya, Japan, July 2021.
Dhruvkumar Patel and Rym Z. Wenkstern. Collaborative collision avoidance for CAVs in unpredictable scenarios. In proceedings of the 3rd IEEE Connected and Automated Vehicles Symposium, CAVS 2020, page 1-6, Virtual, November 2020.
Dhruvkumar Patel and Rym Z. Wenkstern. "16 Connected and Automated Vehicles: Study of Platooning". Vehicles, Drivers, and Safety, edited by Huseyin Abut, Kazuya Takeda, Gerhard Schmidt and John Hansen, page 263-284, Berlin, Boston: De Gruyter, 2020.

Technical Skills:

Languages & Software: Python, Java, C, SQL, Matlab, Tensorflow, Hadoop, Spark.
Web development: Javascript, jQuery, Bootstrap, J2EE.

Web:

LinkedIn: https://www.linkedin.com/in/dhruvkp
Github: https://www.github.com/dhruvkp