Adaptive maneuver planning for autonomous vehicles using behavior tree on Apollo Platform

Mais Jamal[1] and Aleksandr Panov[1,2]

[1] Moscow Institute of Physics and Technology, 141701 Dolgoprudny, Russia
mayssjamal@phystech.edu
[2] Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, 117312 Moscow, Russia
panov.ai@mipt.ru

Abstract. In safety-critical systems such as autonomous driving systems, behavior planning is a significant challenge. The presence of numerous dynamic obstacles makes the driving environment unpredictable. The planning algorithm should be safe, reactive, and adaptable to environmental changes. The paper presents an adaptive maneuver planning algorithm based on an evolving behavior tree created with genetic programming. In addition, we make a technical contribution to the Baidu Apollo autonomous driving platform, allowing the platform to test and develop overtaking maneuver planning algorithms.

Keywords: Maneuver planning · Behavior Trees · Apollo Auto · Self-Driving Cars · Genetic Programming

1 Introduction

In automated driving systems, maneuver planning is a significant challenge. In an unpredictable planning environment, the planner must make the best safe decision. A bad decision endangers not only the passengers' lives, but also the lives of pedestrians and passengers in other nearby vehicles.

Behaviour planners [13, 12] and finite state machines (FSM) [11, 19, 22] are an early approach for intelligent vehicle maneuver planning and decision-making. The possible scenarios are implemented by hand to define its states and transitions between the states, and the rules are based on human experience. However, in complex control systems, such as automated driving systems, there are many rules and traffic situations, making it difficult to implement all world scenarios in the form of finite state machines.

Later approaches [2, 17, 10, 16] are based on probabilistic methods for dealing with uncertainty in decision effects, such as Markov Decision Processes (MDPs), as well as uncertainty in environmental observations, such as Partially Observable Markov Decision Processes (POMDPs). Such methods rely on maximizing the future possible rewards of actions to find the optimal policy. Because this process examines all possible actions, it requires a significant computational load

and is difficult to apply in vehicle behavior planning. A compromise in the number of the states is usually employed to reduce the calculations.

Behavior trees (BT) are another approach that has previously been used in AI games and, more recently, in robotics. BTs are a modular execution processes that switches between a finite set of tasks. A task can be any program or subroutine and it can be either a conditional task or an action task. The control flow nodes define the switching rules between the tasks. Control flow nodes can be either a sequence, which executes all of its children and returns success only if all of its children succeed, or a selector node (fallback node), which executes all of its children and returns success if one of its children succeeds.

Because of their flexibility, reactivity, and modularity, BTs are effective tools for autonomous agent maneuver planning. They are preferred over the FSMs when the number of transitions between the states is significant. As the size of the behavior tree rises proportionally to the complexity of the planning system, several attempts [4, 14, 5, 20] have been made to learn a part or all of its structure from the environment rather than manually constructing the entire tree. The learning approach adapts the behavior tree to the needs of an autonomous vehicle's maneuver planning.

In this paper, we present an adaptive maneuver planning approach based on a learning behavior tree. As an example of maneuver planning, we consider the overtaking maneuver decision-making. We use genetic programming (GP) to learn the entire structure of the BT. The structure is denoted by the character string x. The problem is viewed as an optimization problem, with the GP algorithm searching in the genotype space G for the string $x*$ denoting a behavior tree program for which a fitness function f is optimized:

$$f(x) \rightarrow max : find \overrightarrow{x}^{*} so\ that\ \forall \overrightarrow{x} \in G : f(\overrightarrow{x}) \leq f(\overrightarrow{x}^{*}) = f^{*} \tag{1}$$

The programming language $L = \{F, T\}$ of GP includes two sets: the function set F and the terminal set T. F includes all the possible functions in the program of a BT (control flow nodes). T includes all the possible conditions and actions.

In GP, the genotype space G includes strings representing all BT programs that can be constructed from elements of the programming language L. $P(t) \subset G$ denotes the population's state at time t. The GP's run-time is defined in terms of generations. N individuals $\overrightarrow{x_1}(t), \overrightarrow{x_2}(t), ..., \overrightarrow{x_n}(t)$ make up the population. The GP population evolves from the initial state $P(t = 0)$ by selecting μ parents $(P_p(t))$ for variation and then performing a *variation operation* to them to produce λ-offspring from each parent. The set of offspring is denoted by $P_c(t)$. The variation operation can be either a Crossover or a Mutation. After evaluating the fitness of the new offspring, it is combined with the population to form the new set $P_a(t)$. The new generation $P(t + 1)$ is a subset of $P_a(t)$. Over generations, the GP algorithm seeks for x values that maximize the fitness function f.

To conduct the experiments, we use the Baidu Apollo (open source) autonomous driving platform [1]. The platform has multiple subsystems that give the essential information to the planning subsystem regarding routing, localization, perception, and prediction of surrounding objects. Because of its struc-

ture, as well as its high concurrency and low latency, the platform provides an accurate simulator suitable for developing and testing planning algorithms for autonomous vehicles.

The decision-making process for overtaking maneuvers on the Apollo platform is integrated with motion planning in the planning subsystem. The planning subsystem receives as input: the HD map , ego vehicle location and speed, the surrounding objects, their estimated 3D shape, speed, and a prediction of their future trajectory and heading. In the presence of a slowly moving obstacle in front of the ego vehicle for an extended period of time, the candidates include paths from neighbor lanes to allow lane change if necessary. Then, for each candidate lane, the Piecewise-jerk optimizer [21] is used to optimize the path, followed by optimizing the speed profile of the optimized path. Finally, a trajectory decider will use a cost function to determine which lane to take. With the appearance of dynamic obstacles, the path-speed approach does not guarantee the optimal solution. In response to this limitation, the EM planner is designed to select a lane-change maneuver rather than overtaking (nudging) for high-speed dynamic obstacles.

We have integrated our overtaking planning method, as well as a technical contribution that allows for overtaking high-speed dynamic obstacles, into the Apollo platform.

The remainder of this paper is divided into five sections. Section 2 discusses some recent related works. Section 3 presents the proposed approach of adaptive planning using an evolving behavior tree and describes the simulation environment of Apollo. Section 4 presents the preparation for experiments in Apollo, as well as the experimental results, and analysis. Finally, section 5 concludes with a discussion of future work.

2 Related Work

The works [2, 3, 8] used probabilistic methods of MDPs, POMDPs, or a combination of them with other methods to plan the autonomous vehicle's maneuver. [2] developed a driver assistance maneuver planning framework based on MDP that learns the policy using reinforcement learning (RL) by performing actions and getting a reward. The researchers attempted to compromise the significant computational load by combining offline and online planning. However, the framework still has unresolved issues with computational complexity as the number of objects in the surrounding environment grows. [18] developed an online POMDP algorithm to plan lane changes, which are a part of overtaking maneuvers. The computational complexity was reduced by the proposed algorithm. Because direct state transitions from a lane change left to a lane change right were not considered, the model had only eight states. The algorithm was tested in real-world traffic, but the execution required the driver to confirm the lane changes. [15] modeled the problem of intersection planning as an MDP that learns the policy with RL using what is known as Hierarchical Options (HOMDP). Their results showed an improvement in performance and success rate over POMDP.

Other works like [9, 16] depend on non standard methods for planning and decision-making of lane change maneuver for autonomous vehicles. [9] proposed a behavior planner that selects the appropriate maneuver by creating and evaluating a set of all existing gaps between close dynamic vehicles. A neural network was trained to predict future success based on prior experience and was involved in the evaluation process. The results indicated a high success rate for a lane change. However, the algorithm assumes that other vehicles behind the autonomous one will brake within cautious limits, allowing to change into relatively small gaps. As a result, in high-traffic situations, the algorithm will fail to complete the lane change. On the other hand, [16] proposed a behavior planner to decide whether to stay in the lane or switch to the right/left lane. The proposed planner's model is a state-action spaced deep Q-network that learns through reinforcement learning. The results indicated a relatively high success rate.

Although probabilistic methods and RL [7, 6] have shown good results, they are not reproducible and computationally expensive, necessitating some compromise with the complexity of the planning system. When evolved through genetic programming, behavior trees, on the other hand, are adaptive and reactive. The GP searches for the best fit structure in the space of all possible tree structures without interference from the expert, which may lead to creative solutions. As a result, in this study, we use a behavior tree to plan the overtaking maneuver and genetic programming to find the best-fit structure of the tree.

3 Method

The structure of the overtaking maneuver's behavior tree is learned via genetic programming. The BT determines whether the ego car will overtake the moving vehicle (dynamic obstacle) in front of it from the right or left side, or if it will keep the current lane. The scenario includes several variables: the speed of the front obstacle, the number of dynamic obstacles on both the right and left lanes, their speed and their initial location.

In the Apollo platform, a technical solution for overtaking dynamic obstacles was developed and implemented with the genetic programming to evaluate the individuals (behavior tree structures) in the population.

3.1 Behavior tree primitives

The phenotype is the program version of the genotype. To evaluate the fitness of the BT individual, a mapping between genotype and phenotype is required. A set of all the BT's action, condition, and control nodes defines the phenotypic primitives. *KeepLane*, *SwitchToLeft*, and *SwitchToRight* are the possible actions. A Control node might be a sequence or a selector node. Each neighbor lane is split into three zones with varying lengths of s to reflect occupancy status (Fig. 1). $s(t)$ indicates the safe following-distance and is dependent on the autonomous

vehicle's current speed. The "three-second rule" is used to define it:

$$s(t) = \tau * v(t) \tag{2}$$

where $\tau = 3\ s$. There are two states for each zone: $\{free, occupied\}$. Furthermore, the speed of an occupant obstacle (km/h) is treated as a range and is expressed in six conditions: $[1, 10], [11, 20], [21, 30], [31, 40], [41, 50], [51, 60]$. Table 1 contains the whole list of conditions, actions, and descriptions.

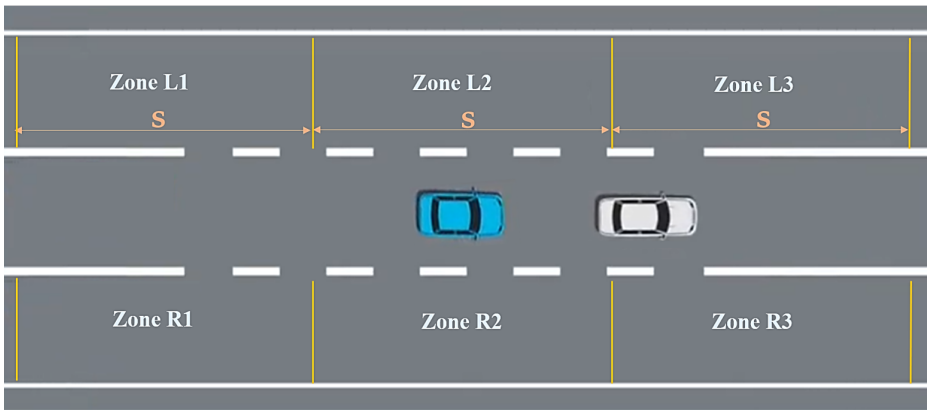

Fig. 1. The zones of the two adjacent lanes. The left lane is divided into three zones L1, L2, and L3. The right lane is also divided into R1, R2, and R3 zones. Each zone has a variable length of s.

The state of the BT can be one of three: SUCCESS, FAIL, or RUNNING. When an action is called for, the RUNNING status is activated. If the tree state is RUNNING for an extended period of time without the action being performed, the status is considered as a FAIL. Adding the RUNNING state to the tree's characteristics, not only excludes the trees that lead to an inapplicable actions, but it also maintains the tree's reactivity when the environmental conditions change during the action.

Characters representing actions and conditions (Table 1) form the terminal set $T = \{c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, X, Y, Z\}$, which maps the behavior tree program (phenotype) to the genotype alphabet. The sequence node is represented by the symbol '&' with two parentheses containing the sub-trees under the sequence node. Similarly, the selector node is represented by the symbol '/' with two parentheses. The function set F is the set of two control functions:$\{\&, /\}$. For instance, the overtaking behavior tree that decides to overtake only when one of the adjacent lanes is clear of obstacles (Fig. 2) is represented by the following string: '/(&(cegY)&(ikmZ)X)'.

Table 1. Primitives of the behavior tree. The possible actions and conditions, their description and the alphabet representing them.

Alphabet	Actions	Description
Y	SwitchToLeft	Switch from the current lane to the left lane.
Z	SwitchToRight	Switch from the current lane to the right lane.
X	KeepLane	Keep going on the current lane.
Alphabet	Conditions	Description
c	L1Free	Is Zone L1 free? (Fig. 1)
d	L1Occ	Is Zone L1 occupied? (Fig. 1)
e	L2Free	Is Zone L2 free? (Fig. 1)
f	L2Occ	Is Zone L2 occupied? (Fig. 1)
g	L3Free	Is Zone L3 free? (Fig. 1)
h	L3Occ	Is Zone L3 occupied? (Fig. 1)
i	R1Free	Is Zone R1 free? (Fig. 1)
j	R1Occ	Is Zone R1 occupied? (Fig. 1)
k	R2Free	Is Zone R1 free? (Fig. 1)
l	R2Occ	Is Zone R2 occupied? (Fig. 1)
m	R3Free	Is Zone R3 free? (Fig. 1)
n	R3Occ	Is Zone R3 occupied? (Fig. 1)
o	ObsSpeed1-10	Is the speed of the occupant obstacle between 1 and 10 km/h?
p	ObsSpeed11-20	Is the speed of the occupant obstacle between 11 and 20 km/h?
q	ObsSpeed21-30	Is the speed of the occupant obstacle between 21 and 30 km/h?
r	ObsSpeed31-40	Is the speed of the occupant obstacle between 31 and 40 km/h?
s	ObsSpeed41-50	Is the speed of the occupant obstacle between 41 and 50 km/h?
t	ObsSpeedMore50	Is the speed of the occupant obstacle more than 50 km/h?
u	EgoSpeed1-10	Is the speed of the Ego vehicle between 1 and 10 km/h?
v	EgoSpeed11-20	Is the speed of the Ego vehicle between 11 and 20 km/h?
w	EgoSpeed21-30	Is the speed of the Ego vehicle between 21 and 30 km/h?
x	EgoSpeed31-40	Is the speed of the Ego vehicle between 31 and 40 km/h?
y	EgoSpeed41-50	Is the speed of the Egovehicle between 41 and 50 km/h?
z	EgoSpeedMore50	Is the speed of the Ego vehicle more than 50 km/h?

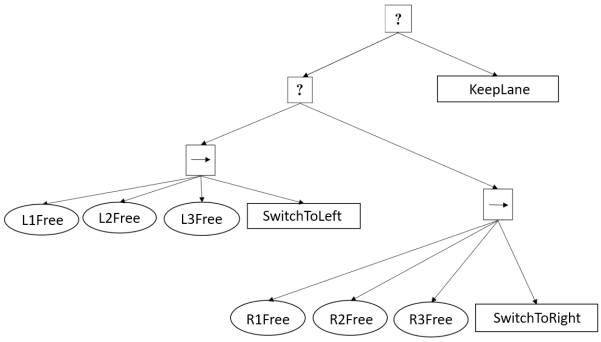

Fig. 2. A simple behavior tree for the overtaking maneuver. The tree begins with the selector node, which checks the SUCCESS state of its children nodes. This causes the second selector node to check its children. The first child (sequence node) evaluates the three conditions of the left lane's zones state; if three free zones L1, L2, L3 exist, the decision is made to switch to the left lane, and the tree returns the state RUNNING. If one of the three conditions is not satisfied, the sequence node returns the state FAIL and the selector node continues to check the second child; if it also returns FAIL, the tree will finally decide to keep lane.

3.2 Genetic programming

To find the optimal structure of the behavior tree, we use a GP execution scheme similar to the one described in [5]. However, for the planning optimization problem, we chose better-fitting parameters (Table 2). The binary tournament selection method is used to select μ parents for variation. This method selects two random individuals and the one with the higher fitness value is chosen and added to the parents' set. Tournament selection allows individuals with poor performance to maintain their genotype, resulting in better exploration of the genotype space and less likelihood of being stuck in a local maximum. λ-offspring are created by performing either a crossover operation or a mutation operation on the selected parents with a probability (Algorithm 1). The variation operations are as follows:

Crossover operation This operation takes two individuals P_1, P_2 from the selected parents $P_p(t)$ and performs a sub-tree swapping to create two new offspring C_1, C_2.

Mutation operation This operation allows an individual to vary in one of three ways with probabilities: addition, mutation, or deletion. In mutation/addition, a random element from the programming language L is added to the tree. In mutation/mutation, a random element from L is mutated to another element in L. In mutation/deletion, a terminal element (condition or action) gets removed from the tree.

The mutation percent is set at 60% to improve the discovery of the searching space and to avoid the rapid growth in the depth of the tree caused by the crossover operation. 70% of the mutation operations are mutation or addition

to explore all the possible actions and conditions while searching for the optimal solution.

Following variation, all offspring ($P_c(t)$) are evaluated and combined with the population $P(t)$ to form $P_a(t)$. Afterwards, e individuals with the highest fitness values are selected from $P_a(t)$ to survive to the next generation (Elitist selection), and the rest of the generation($N - e$) is chosen by the binary tournament selection without repeating any individual. All previously surviving individuals are evaluated again on the new scenario in the next iteration of GP. The following equation combines the previous fitness value associated with the survived individuals with the new fitness value calculated on the new iteration's scenario:

$$f(\overrightarrow{x}(t)) = \alpha * f(\overrightarrow{x}(t-1)) + (1-\alpha) * f(\overrightarrow{x}(t)) \tag{3}$$

where $\alpha = 1/2$. In this manner, a BT program's fitness will be defined as an integration of its fitness over scenarios.

Table 2. Parameters of Genetic Programming.

Parameters of GP	
Population size (N)	20
Parents number(μ)	10
Parents selection method	Binary Tournament
Offspring number(λ)	4
Max depth	6
Crossover percent	40%
Mutation percent	60%
Mutation/ addition percent	40%
Mutation/ deletion percent	30%
Mutation/ mutation percent	30%
Elitist selection percent	10%

Each individual's (BT) fitness in the scenario is determined by two factors: reaching the goal without colliding, and the time it takes to reach the goal following this BT program. If the individual (BT) caused a collision, a negative fitness value will be assigned to it. If the goal is reached without a collision, the fitness is defined as the difference between the time spent following the action *KeepLane* T^{a_0} and the time spent following the tree's output action T^{a_x}:

$$f(x) = T^{a_0} - T^{a_x} \tag{4}$$

As a result, the BT that results to an overtaking and reaching the goal faster without colliding will be assigned a high fitness value. The BT that leads to the *KeepLane* action will be assigned a fitness value close to zero.

3.3 Implementation in Apollo

The Apollo planning module includes two essential planners, *Public Road Planner*, which handles lane follow, junction, and U-sharp turn scenarios, and *Open*

Space Planner, which handles parking scenarios. Every planner initializes a *scenario manager* and updates it every planning cycle. The *scenario manager* specifies the current scenario, processes it, and executes a series of predefined tasks for each scenario. Tasks can either be deciders or optimizers.

Algorithm 1 Genetic Programming

$INPUT : N, MaxGenerations, e(ElitesNumber),$
$Crossover - Percent.$
$OUTPUT : \overrightarrow{x}^* so\ that\ \forall \overrightarrow{x} \in G : f(\overrightarrow{x}) \leq f(\overrightarrow{x}^*) = f^*$
$t = 0$
$initialize : P(t=0) = \{\overrightarrow{x_1}(0), \overrightarrow{x_2}(0), ..., \overrightarrow{x_n}(0)\} \in G$
while $t < MaxGenerations$ **do**
  $evaluate : P(t) : \{f(\overrightarrow{x_1}(t)), f(\overrightarrow{x_2}(t), ..., f(\overrightarrow{x_n}(t))\}$
  $P_p(t) = TournamentSelection(\mu, P(t))$
  $P_c(t) = \emptyset$
  for $i = 1, 2, ..., \mu$ **do**
    $rand = RandNumber(1 - 100)$
    $P_1 = P_p^i(t)$
    if $rand \leq Crossover - Percent$ **then**
      for $k = 1, ..., \lambda/2$ **do**
        $P_2 = RandSelect(P_p(t))$
        $C_1, C_2 = CrossoverOperation(P_1, P_2)$
        $P_c(t) = P_c(t) \cup C_1 \cup C_2$
      end for
    else
      for $k = 1, ..., \lambda$ **do**
        $C = MutationOperation(P_1)$
        $P_c(t) = P_c(t) \cup C$
      end for
    end if
  end for
  $evaluate : P_c(t) : \{f(\overrightarrow{x_{c1}}(t)), f(\overrightarrow{x_{c2}}(t), ..., f(\overrightarrow{x_{cn}}(t))\}$
  $P_a(t) = P_c(t) \cup P(t)$
  $P(t+1) = ElitistSelection(e, P_a(t))$
  $P(t+1) = P(t+1) \cup TournamentSelect(N - e, P_a(t))$
  $t \leftarrow t + 1$
end while

To demonstrate the planning process in Apollo, we consider the planning of a lane-follow scenario in which the planner can make an overtaking decision. The planning process begins with the *Lane Change Decider*, which determines if the road has additional lane available for change if necessary. *Path Lane Borrow Decider* then checks for the existence of a front static long-term blocking obstacle, making the decision to overtake if it is there. The optimal path profile in Frenet is then found using the Jerk path optimizer, followed by the optimal speed profile in Frenet using the Jerk speed optimizer.

The *Path Lane Borrow Decider* in Apollo is designed to make overtaking decisions exclusively for static front obstacles and obstacles with very slow speeds. The proposed behavior tree method is implemented in a BT decider, which is added to the tasks of the Lane-follow scenario in Apollo before the *Path Lane Borrow Decider*. All the primitive conditions and actions of the BT are programmed in the decider as functions. The decider begins by decoding the tree string generated by the GP from the genotype space to the phenotypic space. The decider then runs a recursive tree execution function to perform the behavior tree tasks based on the control flow nodes. For the behavior tree evaluation process, an extra collision check function and a goal-reaching function were also developed.

4 Experiment

4.1 Simulation in Apollo

The source code of Apollo is the pre-version of the Baidu Apollo Open Platform 6. The source code was modified and a decider of the behavior tree was added to the tasks to allow testing of the overtaking maneuver, performance of the behavior tree, and evaluation of it on each planning cycle. To run experiments locally we used an Intel Core i7-9700 CPU:8×3GHz computer with 32 GB RAM, and NVIDIA GeForce RTX 2080 Ti video card. The behavior tree decider runs with a frequency of $\approx$ 10 HZ frequency. The algorithm simulates a random overtaking scenario to evaluate the individuals (BTs) during the GP.

The overtaking scenario includes three lanes on a forward highway road, with the ego vehicle always in the middle lane (Fig. 3). A front obstacle with a starting distance of 20 m exists. The speed of the front obstacles is assigned at random between 2.7 and 5.5 m/s. There are also 5 obstacles placed at random on the neighboring lanes. The neighboring obstacles move at random speeds (between 2.7 and 6.5 m/s). The experiments are run in Apollo's SimControl mode, and the obstacles data is published at a rate of 10 Hz on the Perception channel. Starting from the autonomous agent's initial location, the distance to the goal is 200 m.

4.2 Experiment result and analysis

Running 50 generations of the GP algorithm took nearly 48 hours, which is considered very time-consuming. When compared to the time required for each experiment to evaluate a BT individual, the time required for creating offspring and decoding the BT's string is relatively short and may be neglected. Each evaluation experiment can last between 10 and 70 seconds, depending on the tree's output action and validity. The results of running the GP algorithm for the first 50 generations, starting with a random first population $P(0)$, are shown in Fig. 4. A behavior tree can either be successful or unsuccessful. A tree is considered successful if it achieves its goal without colliding with any of the surrounding obstacles, regardless of the type of output action (KeepLane, SwitchToLeft,

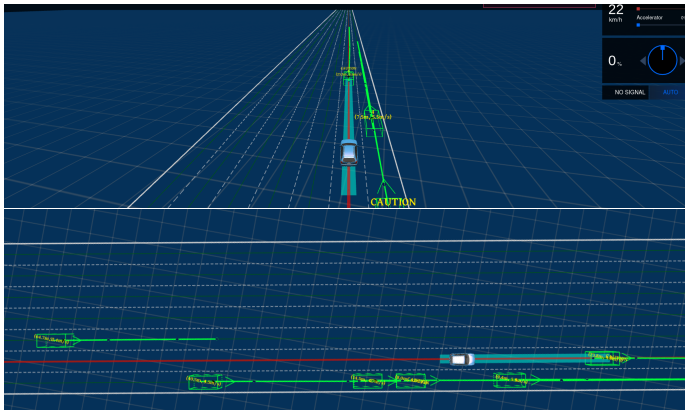

Fig. 3. A random overtaking scenario. The ego vehicle is surrounded by five random dynamic obstacles and one in the front.

SwitchToRight). $S(t) \subseteq P(t)$ denotes the set of successful BTs, $O(t) \subseteq S(t)$ denotes the set of successful BTs with an overtake output action. While a tree is considered an unsuccessful tree in the following cases: the BT output action resulted in a collision, the action run-time has exceeded a predefined maximum limit and the action has not yet been performed, or the BT's execution has ended and no action is running.

$U(t) \subseteq P(t)$ denote the set of unsuccessful BTs and $P(t) = S(t) \cup U(t)$. The results in Fig. 4 show that the number of successful trees increases over generations, while the number of unsuccessful trees decreases significantly. It can be seen that the evolving population has found its way to generate behavior trees with an overtaking action that has been successful in some scenarios. Starting with a random population may cause the searching problem to take longer, but such an algorithm should never end in a local maximum, because the mutation feature and the selection method spread the searching process.

Table 3. A comparison between GP with prior and GP with random $P(0)$.

	GP with prior			**GP with random**		
t	$S(t)\%$	$O(t)\%$	$L(t)\%$	$S(t)\%$	$O(t)\%$	$L(t)\%$
1	80	0	0	5	0	5
5	100	0	5	75	0	20
10	100	0	15	95	0	60
15	95	5	40	95	0	15
20	100	75	35	100	0	60
25	100	90	75	100	0	60
30	100	75	95	100	0	50

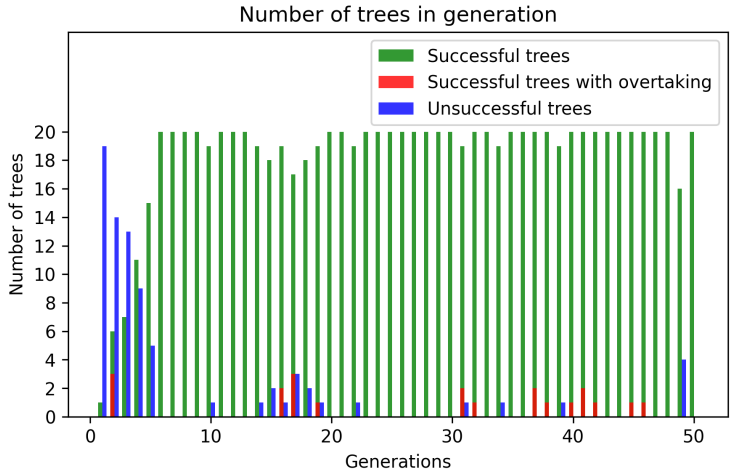

Fig. 4. A bar graph of successful and unsuccessful trees over generations from the first generation to the 50th. The initial population is a randomly generated population.

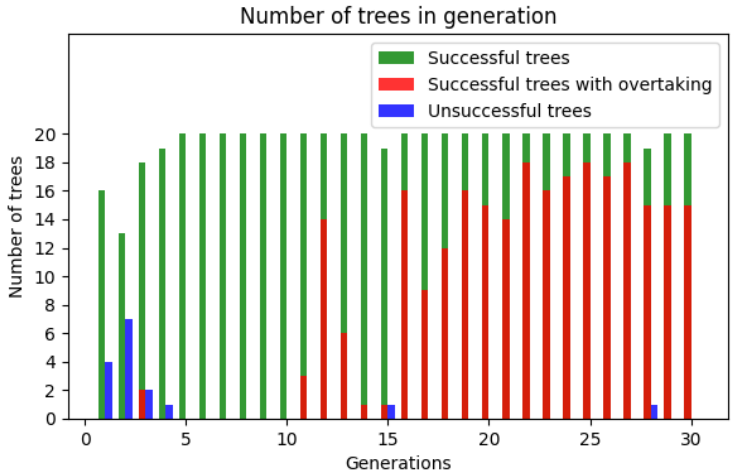

Fig. 5. A bar graph of the number of successful and unsuccessful trees over generations from the first generation to the 30th. The initial population contains one simple effective behavior tree.

Figure 5 shows the results of running the GP algorithm for the first 30 generations, starting from an initial population that contains one simple effective behavior tree (Fig. 2). Results show that the genetic algorithm is dynamically converging to the optimum fitness value. The algorithm optimizes the fitness function over generations, by evolving the trees that output an overtaking action to reach the goal faster, and also keeps the size of the tree reasonable due to the constraint of maximum depth of the behavior tree.

The set of BT individuals with a length of l bigger than 40 symbols (including parentheses) is denoted by $L(t) \subseteq P(t)$. Table 3 presents a percentage comparison between successful trees, overtaking successful trees, and the length of the tree over generations, when GP searching starts with random and with prior.

5 Conclusion and Future Work

Running the proposed algorithm only for the first generations of the GP algorithm yielded promising results for adaptive maneuver planning due to its high flexibility, modularity, and potential. Although the time required to run each experiment in the Apollo platform is a challenge, we can overcome this by running multiple instances of Apollo in parallel to evaluate many behavior tree structures at the same time (Fig. 6). This can be done with high-performance hardware such as supercomputers. This solution will significantly reduce the amount of time spent searching for the optimal BT structure. Another improvement to the algorithm can be made by employing a more effective evolutionary algorithm, expanding the environmental conditions, and including information from Apollo's Prediction module.

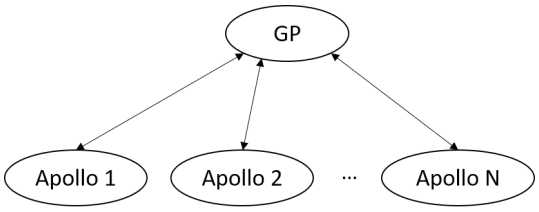

Fig. 6. Multi-instance architecture of GP

Acknowledgements. The reported study was supported by RFBR, research Project No. 18-29-22027.

References

1. Baidu Apollo team (2017), Apollo: Open Source Autonomous Driving, howpublished = https://github.com/apolloauto/apollo, note = Accessed: 2019-02-11

2. Brechtel, S., Gindele, T., Dillmann, R.: Probabilistic mdp-behavior planning for cars. In: 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC). pp. 1537–1542. IEEE (2011)
3. Brechtel, S., Gindele, T., Dillmann, R.: Probabilistic decision-making under uncertainty for autonomous driving using continuous pomdps. In: 17th International IEEE Conference on Intelligent Transportation Systems (ITSC). pp. 392–399 (2014). https://doi.org/10.1109/ITSC.2014.6957722
4. Fu, Y., Qin, L., Yin, Q.: A reinforcement learning behavior tree framework for game ai. In: 2016 International Conference on Economics, Social Science, Arts, Education and Management Engineering. pp. 573–579. Atlantis Press (2016)
5. Iovino, M., Styrud, J., Falco, P., Smith, C.: Learning behavior trees with genetic programming in unpredictable environments. arXiv preprint arXiv:2011.03252 (2020)
6. Ivanov, D., Panov, A.I.: Application of Reinforcement Learning in Open Space Planner for Apollo Auto. In: Kovalev, S., Tarassov, V., Snasel, V., Sukhanov, A. (eds.) Proceedings of the Fifth International Scientific Conference "Intelligent Information Technologies for Industry" (IITI'21). IITI'21 2021. Advances in Intelligent Systems and Computing. p. (In Press). Springer (2021)
7. Kiselev, G., Panov, A.I.: Q-learning of Spatial Actions for Hierarchical Planner of Cognitive Agents. In: Ronzhin, A., Rigoll, G., Meshcheryakov, R. (eds.) Interactive Collaborative Robotics. ICR 2020. Lecture Notes in Computer Science. vol. 12336, pp. 160–169. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-60337-3_16, https://link.springer.com/chapter/10.1007/978-3-030-60337-3_16 https://www.scopus.com/record/display.uri?eid=2-s2.0-85092940528&origin=resultslist
8. Martinson, M., Skrynnik, A., Panov, A.I.: Navigating Autonomous Vehicle at the Road Intersection Simulator with Reinforcement Learning. In: Kuznetsov, S.O., Panov, A.I., Yakovlev, K.S. (eds.) Artificial Intelligence. RCAI 2020. Lecture Notes in Computer Science. vol. 12412, pp. 71–84. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-59535-7_6, https://link.springer.com/chapter/10.1007/978-3-030-59535-7_6 https://www.scopus.com/record/display.uri?eid=2-s2.0-85092200949&origin=resultslist
9. Menéndez-Romero, C., Winkler, F., Dornhege, C., Burgard, W.: Maneuver planning for highly automated vehicles. In: 2017 IEEE Intelligent Vehicles Symposium (IV). pp. 1458–1464. IEEE (2017)
10. Mirchevska, B., Pek, C., Werling, M., Althoff, M., Boedecker, J.: High-level decision making for safe and reasonable autonomous lane changing using reinforcement learning. In: 2018 21st International Conference on Intelligent Transportation Systems (ITSC). pp. 2156–2162. IEEE (2018)
11. Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., et al.: Junior: The stanford entry in the urban challenge. Journal of field Robotics **25**(9), 569–597 (2008)
12. Osipov, G.S., Panov, A.I.: Rational behaviour planning of cognitive semiotic agent in dynamic environment. Scientific and Technical Information Processing **48**(6), (In press) (2021)
13. Panov, A.I.: Goal Setting and Behavior Planning for Cognitive Agents. Scientific and Technical Information Processing **46**(6), 404–415 (2019). https://doi.org/10.3103/S0147688219060066

14. Pereira, R.d.P., Engel, P.M.: A framework for constrained and adaptive behavior-based agents. arXiv preprint arXiv:1506.02312 (2015)
15. Qiao, Z., Muelling, K., Dolan, J., Palanisamy, P., Mudalige, P.: Pomdp and hierarchical options mdp with continuous actions for autonomous driving at intersections. In: 2018 21st International Conference on Intelligent Transportation Systems (ITSC). pp. 2377–2382. IEEE (2018)
16. Rezaee, K., Yadmellat, P., Nosrati, M.S., Abolfathi, E.A., Elmahgiubi, M., Luo, J.: Multi-lane cruising using hierarchical planning and reinforcement learning. In: 2019 IEEE Intelligent Transportation Systems Conference (ITSC). pp. 1800–1806. IEEE (2019)
17. Ulbrich, S., Maurer, M.: Probabilistic online pomdp decision making for lane changes in fully automated driving. In: 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013). pp. 2063–2067. IEEE (2013)
18. Ulbrich, S., Maurer, M.: Probabilistic online pomdp decision making for lane changes in fully automated driving. In: 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013). pp. 2063–2067 (2013). https://doi.org/10.1109/ITSC.2013.6728533
19. Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M., Dolan, J., Duggins, D., Galatali, T., Geyer, C., et al.: Autonomous driving in urban environments: Boss and the urban challenge. Journal of Field Robotics **25**(8), 425–466 (2008)
20. Zhang, Q., Yao, J., Yin, Q., Zha, Y.: Learning behavior trees for autonomous agents with hybrid constraints evolution. Applied Sciences **8**(7), 1077 (2018)
21. Zhang, Y., Sun, H., Zhou, J., Pan, J., Hu, J., Miao, J.: Optimal vehicle path planning using quadratic optimization for baidu apollo open platform. In: 2020 IEEE Intelligent Vehicles Symposium (IV). pp. 978–984 (2020). https://doi.org/10.1109/IV47402.2020.9304787
22. Ziegler, J., Bender, P., Schreiber, M., Lategahn, H., Strauss, T., Stiller, C., Dang, T., Franke, U., Appenrodt, N., Keller, C.G., et al.: Making bertha drive—an autonomous journey on a historic route. IEEE Intelligent transportation systems magazine **6**(2), 8–20 (2014)