Discovering Human-like Driving Strategies with a Multiobjective Optimization Algorithm

Erik Dovgan

Faculty of Computer and Information Science, University of Ljubljana, Slovenia
Department of Intelligent Systems, Jožef Stefan Institute, Slovenia

February 2018

erik.dovgan@ijs.si, erik.dovgan@fri.uni-lj.si
# Contents

1. Introduction .................................................. 3

2. The MOHDS algorithm ....................................... 5

3. The SCANeR Autonomous Driving algorithm ............... 6

4. The *Autonomous vehicle* package .......................... 7
   4.1 The *av_control* script ................................... 7
   4.2 The *av_functions* script .................................. 7
   4.3 The *av_model_data* script .............................. 7
   4.4 The *av_scenario_data* script ......................... 7
   4.5 The *av_simulator* script ................................ 8
   4.6 The *av_status* script .................................. 8

5. The *Simulation* package .................................. 9
   5.1 The *ts_io* script ....................................... 9
   5.2 The *av_scenario_data* script ......................... 9
   5.3 The *ts_simulation_control* script .................... 9
   5.4 The *av_simulation_data* script ..................... 9
   5.5 The *av_statistics_data* script ...................... 10
   5.6 The *ts_traffic_vehicle_control* script ............. 10

6. The *Optimization* package .................................. 11
   6.1 The *human_driving_data* script ....................... 11
   6.2 The *human_driving_functions* script ................. 11
   6.3 The *hypervolume* script ................................ 11
   6.4 The *io* script ......................................... 12
   6.5 The *mo_data* script ................................... 12
   6.6 The *util_random* script ................................ 12
   6.7 The *utils* script ..................................... 13

7. The *SCANeR* package ...................................... 14
   7.1 The *autonomous_vehicle* script ....................... 14
   7.2 The *autonomous_vehicle_data* script ................. 14

8. The *Parameters* package .................................... 15
   8.1 The *vehicle_parameters* script ....................... 15
   8.2 The *simulation_parameters* script ................... 15
   8.3 The *optimization_parameters* script ................. 15
   8.4 The *scaner_simulation_parameters* script .......... 15

This work was financially supported by the Slovenian Research Agency under the research project "Multiobjective discovery of driving strategies for autonomous vehicles", funding no. Z2-7581.
1 Introduction

This report describes the implementation of the Multiobjective Optimization algorithm for discovering Human-like Driving Strategies (MOHDS). This algorithm has been developed in the project “Multiobjective discovery of driving strategies for autonomous vehicles” [3, 4], which was financially supported by the Slovenian Research Agency [1]. In addition, this algorithm has been presented at the Genetic and Evolutionary Computation Conference (GECCO) 2017 [5].

MOHDS is a two-level multiobjective optimization algorithm that mimics human driving behavior and at the same time optimizes the traveling time, the fuel consumption, and the similarity with human driving. The lower-level algorithm consists of a set of mathematical models that mimic human driving behavior. The algorithm observes the vehicle state and the state of neighboring vehicles, and selects the best control actions. The upper-level algorithm is a multiobjective optimization algorithm based on DEMO [7] and NSGA-II [2] that searches for the best values of the input parameters for the lower-level algorithm by optimizing the traveling time, the fuel consumption, and the similarity with human driving [5].

This report describes the source code of the MOHDS algorithm, which is available on the project web site [4]. The code consists of five packages and two main algorithms. The main algorithms are the MOHDS algorithm and the SCANeR Autonomous Driving algorithm. The MOHDS algorithm searches for driving strategies and optimizes the objectives mentioned above. The discovered driving strategies are then integrated in the SCANeR Simulation Environment [6] with the SCANeR Autonomous Driving algorithm. These algorithms use the code from the following packages as follows:

- **The MOHDS algorithm**
  - **Autonomous vehicle**
    This package simulates the behavior of the autonomous vehicle, its decisions with respect to the selected driving strategy, and the execution of the decisions.
  - **Simulation**
    This package sets up and controls the driving simulation, including the simulation of the traffic vehicles.
  - **Optimization**
    This package optimizes the driving strategies. To this end, it evaluates the driving strategies by using the **Autonomous vehicle** package to simulate the driving within the driving simulation, provided by the **Simulation** package.

- **The SCANeR Autonomous Driving algorithm**
  - **SCANeR**
    This package controls the autonomous vehicle by taking into account the driving context provided by the SCANeR Simulation Environment.

- **Algorithm parameters**

---

This work was financially supported by the Slovenian Research Agency under the research project “Multiobjective discovery of driving strategies for autonomous vehicles”, funding no. Z2-7581.
– Parameters

This package stores the parameters of the both above mentioned algorithms.

The code is written in Python. The following sections describe the Python scripts stored in these packages.
2 The MOHDS algorithm

The MOHDS algorithm is implemented in the mohds script. This is an evolutionary algorithm based on the DEMO [7] and NSGA-II [2]. It consists of the following functions:

- **evolution**: performs the evolution that improves a population of driving strategies through generations using generic operators
- **init_min_max_objectives**: initializes the minimum and maximum values of the objectives
- **init_population**: randomly initializes the population
- **create_rand_chromosome**: creates a randomly initialized driving strategy
- **evaluate**: evaluates a driving strategy
- **truncate_population**: returns the best driving strategies in the population
- **create_chromosome**: create a new driving strategy from existing driving strategies using genetic operators
- **make_feasible_chromosome**: makes a driving strategy feasible
- **set_feasible**: makes a datum in the driving strategy feasible

For more details on the MOHDS algorithm see [5].

---

This work was financially supported by the Slovenian Research Agency under the research project “Multiobjective discovery of driving strategies for autonomous vehicles”, funding no. Z2-7581.
3 The SCANeR Autonomous Driving algorithm

The SCANeR Autonomous Driving algorithm is implemented in the `scaner_autonomous_driving` script. It enables to control a SCANeR vehicle [6] with the selected driving strategy. To this end, it is implemented as a SCANeR module and requires to set the path to the SCANeR DLL library and SCANeR Python interface [6]. The SCANeR vehicle is controlled by firstly reading the data of vehicles from SCANeR, secondly calculating the acceleration and lane changing, and finally sending the control data to SCANeR.

This algorithm is to be included in SCANeR as follows. A new process has to be added to the simulation. This process should have the following data:

- Executable: "<absolute path to Python>"
- Process Name: "<user-defined name of the new process>"
- Extra Argument: "-u <absolute path to scaner_autonomous_driving.py> -c <SCANeR configuration name> -p <user-defined name of the new process> -f <user-defined frequency>"

This work was financially supported by the Slovenian Research Agency under the research project “Multiobjective discovery of driving strategies for autonomous vehicles”, funding no. Z2-7581.
4 The Autonomous vehicle package

4.1 The av_control script

This script determines the context of the vehicle driving and controls the vehicle behavior by executing the selected driving strategy. It contains the following functions:

- \texttt{update\_time}: updates the simulation time
- \texttt{determine\_segment}: determines the route segment
- \texttt{determine\_context}: determines the data of the traffic vehicles
- \texttt{make\_decision}: controls the vehicle behavior
- \texttt{set\_acceleration}: calculates the vehicle acceleration
- \texttt{calculate\_velocity\_multiply}: determines the multiplication coefficient for the target velocity by taking into account the route inclinations and turnings
- \texttt{calculate\_max\_a}: determines the maximum allowed acceleration
- \texttt{manage\_lane\_change}: determines when to change the lane
- \texttt{change\_lane}: executes lane changing

4.2 The av_functions script

This script initializes the vehicle and scenario data. It contains the following functions:

- \texttt{set\_scenario\_data}: initializes the scenario data
- \texttt{set\_model\_parameters}: initializes the data of vehicle driving models
- \texttt{calculate\_pos\_x\_rear}: calculates the rear position of the vehicle
- \texttt{init\_autonomous\_vehicle}: initializes the vehicle status
- \texttt{update\_status}: updates the vehicle status

4.3 The av_model_data script

This script stores the data of the vehicle models.

4.4 The av_scenario_data script

This script stores the scenario data.
4.5 The \textit{av\_simulator} script

This script simulates the vehicle behavior. It contains the following functions:

- \texttt{calculate\_feasible\_acceleration}: calculates the maximum feasible acceleration
- \texttt{get\_max\_torque}: calculates maximum engine torque
- \texttt{get\_fuel\_consumption}: calculates fuel consumption
- \texttt{get\_fuel\_consumption\_g\_per\_kwh}: calculates fuel consumption in grams per kilo Watt hours

4.6 The \textit{av\_status} script

This script stores the status of the vehicle.

\textit{This work was financially supported by the Slovenian Research Agency under the research project "Multiobjective discovery of driving strategies for autonomous vehicles", funding no. Z2-7581.}
5 The Simulation package

5.1 The ts_io script

This script provides the descriptions of the simulation. It contains the following functions:

- `get_simulation_status`: provides the status of all the vehicles
- `get_file_header`: provides the header of the status of all the vehicles
- `get_vehicle_file_header`: provides the header of the detailed status of the autonomous vehicle
- `get_vehicle_status`: provides the detailed status of the autonomous vehicle

5.2 The av_scenario_data script

This script stores the data on four scenarios and on the selected scenario.

5.3 The ts_simulation_control script

This script controls the simulation. It contains the following functions:

- `evaluate_driving`: performs the vehicle driving and stores the driving data in files
- `init_statistics`: initializes the driving statistics
- `check_feasibility`: checks the driving feasibility
- `init_traffic_vehicles`: initializes the data of traffic vehicles
- `init_simulation_data`: initializes the simulation data
- `calculate_min_traveling_time`: calculates the minimum traveling time
- `calculate_min_max_traveling_time_and_fuel_consumption`: calculates/estimates the minimum and maximum traveling time and fuel consumption
- `update_simulation`: executes one simulation step
- `update_status`: updates vehicle status
- `store_driving_data`: stores the driving data history
- `check_collisions`: checks for collisions
- `evaluate_scenario`: evaluates a scenario

5.4 The av_simulation_data script

This script stores the simulation data.

This work was financially supported by the Slovenian Research Agency under the research project “Multiobjective discovery of driving strategies for autonomous vehicles”, funding no. Z2-7581.
5.5 The **av_statistics_data** script

This script stores the driving statistics data.

5.6 The **ts_traffic_vehicle_control** script

This script controls the traffic vehicles. It contains the following functions:

- *set_target_velocity*: sets the target velocity
- *get_main_target_velocity*: returns the target velocity on the main lane
- *calculate_acceleration*: calculates the acceleration
- *calculate_acceleration_from_first_order_function*: calculates a smooth acceleration change with respect to the current and target velocities
- *check_traffic_vehicle_positions*: checks and updates vehicle position
- *check_negative_velocity*: checks whether the velocity is negative
- *calculate_pos_x_rear*: calculates the vehicle rear position

This work was financially supported by the Slovenian Research Agency under the research project “Multiobjective discovery of driving strategies for autonomous vehicles”, funding no. Z2-7581.
6 The Optimization package

6.1 The human_driving_data script

This script provides the header of human driving data, and the indexes of data that are to be used when evaluating the similarity with human driving.

6.2 The human_driving_functions script

This script calculates the similarity between the driving data of driving strategy and the driving data of humans. It contains the following functions:

- `calculate_human_likeness`: calculates the similarity between the driving data of driving strategy and the driving data of humans
- `calculate_diff_with_user`: calculates the similarity between the driving data of driving strategy and the driving data of one driver
- `interpolate_data`: interpolates the data
- `interpolate`: interpolates a datum
- `normalized_rmse`: normalizes the error
- `init_human_likeness_calculation`: reads human driving data
- `process_original_files`: processes the original human driving files to be ready for the similarity calculation
- `get_objective_values`: simulates the driving from the original human driving files to obtain the objective values

6.3 The hypervolume script

This script calculates the hypervolume [8] of a set of driving strategies. It contains the following functions:

- `calculate_hypervolume`: calculates the hypervolume
- `get_front`: returns the front of driving strategies
- `get_normalized_front`: normalizes the front
- `get_inverted_front`: inverts the front
- `hypervolume_calculation`: calculates the hypervolume from the front
- `filter_nondominated_set`: filters the nondominated set of solutions
- `surface_unchanged_to`: returns the minimum value of the objective on the front

This work was financially supported by the Slovenian Research Agency under the research project “Multiobjective discovery of driving strategies for autonomous vehicles”, funding no. Z2-7581.
• *reduce_nondominated_set*: sorts the front
• *dominates*: determines the domination relation between two driving strategies
• *swap*: swaps the driving strategies on the front

### 6.4 The io script

This script stores the optimization data in files. It contains the following functions:

- *init*: initializes folders and files
- *get_solution_file_name*: returns the file name of the driving strategy history file
- *save_parameters*: saves the optimization parameters
- *save_population_data*: saves the data of the processed driving strategies
- *copy_front*: copies the history data of a set of driving strategies
- *get_chromosome_description*: returns a description of a driving strategy
- *get_chromosome_values*: returns the driving strategy objective values
- *get_chromosome_id_header*: returns the header of the driving strategy data
- *get_chromosome_id_values*: returns the driving strategy data
- *get_chromosome_short_description*: returns a short description of driving strategy
- *get_archive_header*: returns the header of the driving strategy archive
- *save_chromosome_to_archive*: stores the driving strategy in archive

### 6.5 The mo_data script

This script stores the data of the optimization, including the discovered driving strategies.

### 6.6 The util_random script

This script provides a custom random number generator. It contains the following functions:

- *get_rand_double*: returns a random double value
- *rand_shuffle*: shuffles the data
- *get_rand_int*: returns a random integer value
- *get_rand_state*: returns the state of the random generator
- *init*: initializes the random generator

This work was financially supported by the Slovenian Research Agency under the research project “Multiobjective discovery of driving strategies for autonomous vehicles”, funding no. Z2-7581.
6.7 The *utils* script

This script provides the multiobjective functions [2] for the multiobjective optimization algorithm. It contains the following functions:

- *find_best_indexes*: returns the indexes of the best driving strategies
- *fast_nondominated_sort*: performs fast nondominated sort
- *dominates*: provides the domination relation between two driving strategies
- *calculate_crowding_distance*: calculates the crowding distance
- *sort_front*: sorts the fronts of driving strategies
- *get_highest_crowding_distances*: returns the highest crowding distance on the front

This work was financially supported by the Slovenian Research Agency under the research project “Multiobjective discovery of driving strategies for autonomous vehicles”, funding no. Z2-7581.
7 The SCANeR package

7.1 The autonomous_vehicle script

This script calculates the control actions for the SCANeR autonomous vehicle [6]. It contains the following functions:

- `scaner_format_string`: formats the strings to be compatible with the SCANeR interfaces
- `scaner_format_pointer`: formats the pointers to be compatible with the SCANeR interfaces
- `update_time`: updates the simulation time
- `determine_segment`: determines the route segment
- `determine_context`: determines the data of the vehicles
- `calculate_acceleration`: calculates the vehicle acceleration
- `calculate_velocity_multiply`: determines the multiplication coefficient for the target velocity by taking into account the route inclinations and turnings
- `calculate_max_a`: determines the maximum allowed acceleration
- `manage_lane_change`: determines when to change the lane
- `change_lane`: executes lane changing

7.2 The autonomous_vehicle_data script

This script stores the data of the autonomous vehicle and the context data.
8 The **Parameters** package

8.1 The *vehicle_parameters* script

This script stores the parameters of the autonomous vehicle, such as vehicle length, gear ratios, maximum torque, specific fuel consumption, etc.

8.2 The *simulation_parameters* script

This script stores the parameters of the driving simulation.

8.3 The *optimization_parameters* script

This script stores the parameters of the optimization algorithm, including the paths for the optimization result files and path of the human driving data.

8.4 The *scanner_simulation_parameters* script

This script stores the parameters of the SCANeR autonomous vehicle, including the paths to the SCANeR DLL and SCANeR Python interfaces, and the selected driving strategy.

This work was financially supported by the Slovenian Research Agency under the research project “Multiobjective discovery of driving strategies for autonomous vehicles”, funding no. Z2-7581.
References


This work was financially supported by the Slovenian Research Agency under the research project “Multiobjective discovery of driving strategies for autonomous vehicles”, funding no. Z2-7581.