Adaptive Intelligent Vehicle Modules for Tactical Driving

Rahul Sukthankar, John Hancock, Shumeet Baluja, Dean Pomerleau, Charles Thorpe
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3891
Fax: 1-412-268-5571
E-mail: {rahuls|jhancock|baluja|pomerlea|cet}@cs.cmu.edu

Abstract

Intelligent vehicles must make real-time tactical level decisions to drive in mixed traffic environments. SAPIENT is a reasoning system that combines high-level task goals with low-level sensor constraints to control simulated and (ultimately) real vehicles like the Carnegie Mellon Navlab robot vans.

SAPIENT consists of a number of reasoning modules whose outputs are combined using a voting scheme. The behavior of these modules is directly dependent on a large number of parameters both internal and external to the modules. Without carefully setting these parameters, it is difficult to assess whether the reasoning modules can interact correctly; furthermore, selecting good values for these parameters manually is tedious and error-prone. We use an evolutionary algorithm, termed Population-Based Incremental Learning, to automatically set each module's parameters. This allows us to determine whether the combination of chosen modules is well suited for the desired task, enables the rapid integration of new modules into existing SAPIENT configurations, and provides a painless way to find good parameter settings.

Key Words

1. Adaptive agents
2. Intelligent vehicles
3. Tactical reasoning
4. Evolutionary algorithms
5. Simulation

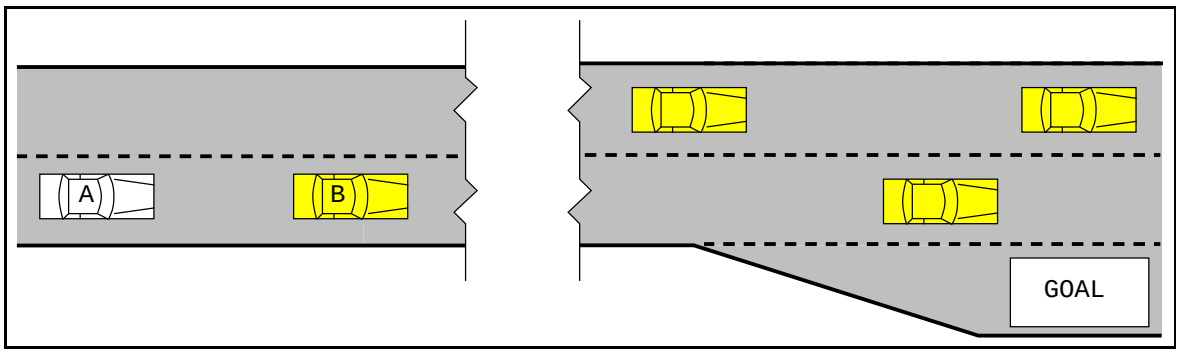

Figure 1: Car A is approaching its desired exit behind a slow vehicle B. Should Car A attempt to pass?

1. Introduction

The task of driving can be characterized as consisting of three levels: strategic, tactical and operational [10]. At the highest (strategic) level, a route is planned and goals are determined; at the intermediate (tactical) level, maneuvers are selected to achieve short-term objectives — such as deciding whether to pass a blocking vehicle; and at the lowest level, these maneuvers are translated into control operations.

Mobile robot research has successfully addressed the three levels to different degrees. Strategic level planners [15, 21] have advanced from research projects to commercial products. The operational level has been investigated for many decades, resulting in systems that range from semi-autonomous vehicle control [6, 9] to autonomous driving in a variety of situations [5, 16, 11]. Substantial progress in autonomous navigation in simulated domains has also been reported in recent years [14, 13, 4, 12]. However, the decisions required at the tactical level are difficult and a general solution remains elusive.

Consider the typical tactical decision scenario depicted in Figure 1: Our vehicle (A) is in the right lane of a divided highway, approaching the chosen exit. Unfortunately, a slow car (B) blocks our lane, preventing us from moving at our preferred velocity. Our desire to pass the slow car conflicts with our reluctance to miss the exit. The correct decision in this case depends not only on the distance to the exit, but also on the traffic configuration in the area. Even if the distance to the exit is sufficient for a pass, there may be no suitable gaps in the right lane ahead before the exit. SAPIENT, described in Section 3, is a collection of intelligent vehicle algorithms designed to drive the Carnegie Mellon Navlab [20, 7] in situations similar to the given scenario. SAPIENT has a distributed architecture which enables researchers to quickly add new reasoning modules to an existing configuration, but it does not address the problem of reconfiguring the parameters in the new system. We present an evolutionary algorithm, Population Based Incremental Learning (PBIL) that automatically searches this parameter space and learns to drive vehicles in traffic.

2. SHIVA

Simulation is essential in developing intelligent vehicle systems because testing new algorithms in human traffic is risky and potentially disastrous. SHIVA[1] (Simulated Highways for Intelligent Vehicle Algorithms) [19, 18] is a kinematic micro-simulation of vehicles moving and interacting on a user-defined stretch of roadway that models the elements of the tactical driving domain most useful to intelligent vehicle designers. The vehicles can be equipped with simulated human drivers as well as sensors and algorithms for automated control. These algorithms influence the vehicles' motion through simulated commands to the accelerator, brake and steering wheel. SHIVA's user interface provides facilities for visualizing and influencing the interactions between vehicles (See Figure 2). The internal structure of the simulator is

[1]For more information, graphics, and a 3-D walk-through, see <http://www.cs.cmu.edu/~rahuls/shiva.html>

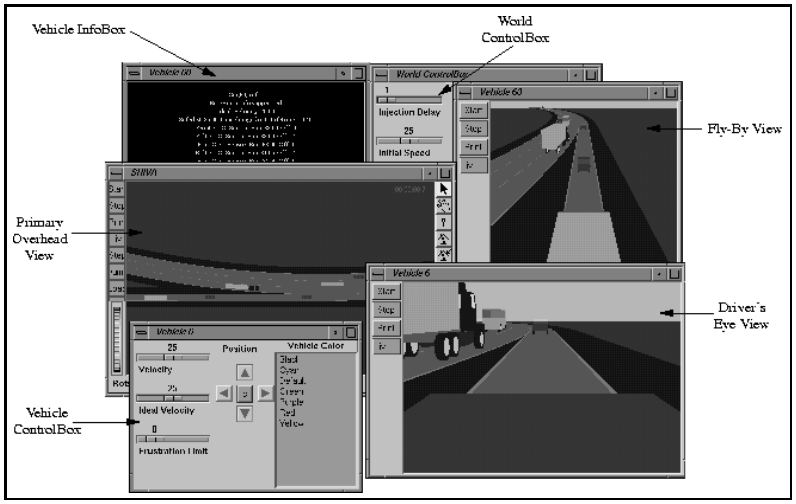

Figure 2: SHIVA: A design and simulation tool for developing intelligent vehicle algorithms.

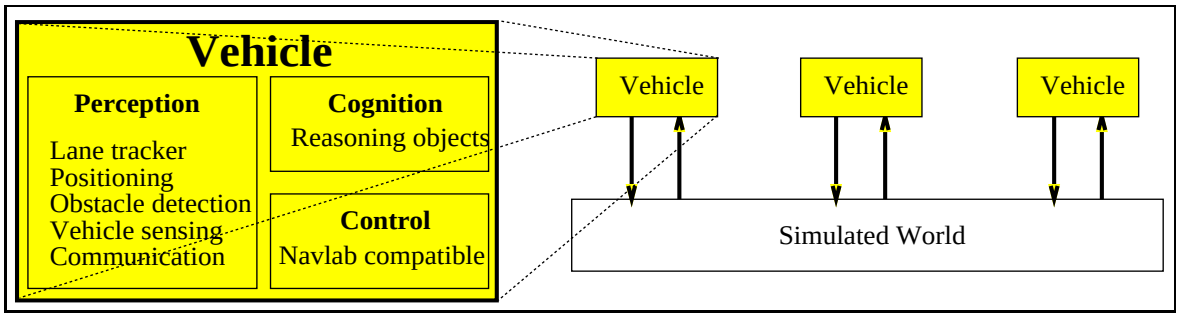

Figure 3: Each vehicle is composed of three subsystems which interact with the simulated world.

comprehensively covered in [19] and details of the design tools may be found in [18].

SHIVA's architecture is open-ended, enabling researchers to simulate interactions between a variety of vehicle configurations. All vehicles can be functionally represented as consisting of three subsystems: perception, cognition and control (See Figure 3).

2.1. Perception

The perception subsystem consists of a suite of functional sensors (e.g. GPS, range-sensors, lane-trackers), whose outputs are similar to real perception modules implemented on the Navlab vehicles. SHIVA vehicles use these sensors to get information about the road geometry and surrounding traffic. Vehicles may control the sensors directly, activating and panning the sensors as needed, encouraging active perception. Some sensors also model occlusion and noise, forcing cognition routines to be realistic in their input assumptions. Two perception components are particularly relevant to this paper: the lane tracker, and the car tracker.

The lane tracker assumes a pure-pursuit[22] model of road-following. This means that the lane tracker suggests a steering arc that will bring the vehicle to the center of the lane after traveling a (velocity dependent) *lookahead distance*. The lane tracker may also be directed to steer the vehicle towards an arbitrary lateral offset on the road. Thus, lane changing is implemented by smoothly varying the lateral position of the pure-pursuit point from the center of one lane to the center of the desired adjacent lane[8]. It is important to note that the actual lateral offset of the vehicle always lags the current position of its pure-pursuit point.

Car tracking is a two step process. In the first phase, the sensor determines the nearest visible vehicle in its range and field of view. In the second, the sensed vehicle's position is transformed into *road coordinates* (i.e.

relative lateral and longitudinal offsets). This allows the tactical reasoning algorithms to remain invariant over changes in road curvature. Car trackers scanning different areas of the road (e.g. front-right, rear-right) are activated as needed during tactical maneuvers to provide relevant information about surrounding traffic.

2.2. Cognition

While a variety of cognition modules have been developed in SHIVA, this paper is only concerned with two types: rule-based reasoning and SAPIENT. The rule-based reasoning module is implemented as a monolithic decision tree. An internal state reflects the current mode of operation (lane changing, lane tracking, seeking an exit etc.) and hand-crafted rules are used to generate actions (steering command and velocity changes) and state transitions. While this system performs well on many scenarios, it suffers from three main disadvantages: 1) modification of the rules is difficult since a small change in desired behavior can require many non-local modifications; 2) hand-coded rules perform poorly in unanticipated situations; 3) implementing new features requires one to consider an exponential number of interactions with existing rules. Similar problems were reported by Cremer *et al* [4] in their initial state-machine implementation for scenario control. To address some of these problems, we have developed a distributed reasoning architecture, SAPIENT, which is discussed more fully in Section 3.

2.3. Control

The control subsystem is compatible with the controller available on the Carnegie Mellon Navlab II robot testbed vehicle, and only allows vehicles to control desired velocity and steering curvature. Denying control over acceleration prevents simulated vehicles from performing operations such as platooning, but ensures that systems developed in simulation can be directly ported onto existing hardware.

3. SAPIENT

SAPIENT (Situational Awareness Planner Implementing Effective Navigation in Traffic) [17, 3] is a reasoning system designed to solve tactical driving problems. To overcome deficiencies with the monolithic reasoning systems described in Section 2.2, SAPIENT partitions the driving task into many independent aspects, each one represented by an independent agent known as a *reasoning object* (See Section 3.1).

3.1. Reasoning Objects

Wherever possible, each *reasoning object* represents a physical entity relevant to the driving task (e.g. car ahead, upcoming exit). Similarly, different aspects of the vehicle's self-state (e.g. how velocity compares to desired velocity) are also represented as individual reasoning objects. Every reasoning object takes inputs from one or more sensors (e.g. the reasoning object for the vehicle ahead monitors forward-facing car tracking sensors).

Each reasoning object tracks relevant attributes of the appropriate entity. Some aspects of the tactical situation can be represented using stateless models of the entity (e.g. speed limits) while others require information about the past (e.g. lane changing). Reasoning objects do not communicate with each other, and are activated and destroyed in response to sensed events and higher level commands. For example, the `Exit Object` activates only when the desired exit is nearby, and `Sensed Car Objects` are destroyed when the vehicle being tracked moves out of the region of interest. Each reasoning object examines the repercussions of each *action* (See Section 3.2) as it would affect the appropriate entity. Thus an `Exit Object` analyzes a possible right lane change only in terms of its impact on the chance of making the desired exit, and ignores the possible interactions with vehicles in the right lane (this is taken care of by other reasoning objects). Every reasoning object then presents its recommendations about the desirability of each

proposed maneuver. For this to work, all reasoning objects must share a common output representation. In SAPIENT, every object votes over a predetermined set of actions.

3.2. Actions

At the tactical level, all actions have a *longitudinal* and a *lateral* component. This choice of coordinate frames allows reasoning objects to be invariant of the underlying road geometry as far as possible. Intuitively, longitudinal commands correspond to speeding up or braking, while lateral commands map to lane changes. More complex maneuvers are created by combining these basic actions. Since the scales in the two dimensions vary widely, longitudinal commands are interpreted as velocity changes, whereas lateral commands correspond to changes in displacement. Thus the null action represents maintaining speed in the current lane.

Tactical maneuvers (such as lane changing) are composed of a sequence of actions, which when concatenated, produce the desired change. Each reasoning object independently maintains the state required to complete the maneuver, and while this decision complicates the internals of certain reasoning objects, it also enables SAPIENT to easily abort maneuvers during an emergency. For example, a left lane change in progress can be aborted if the front-left sensor reports a sudden decrease in velocity of its tracked vehicle.

Reasoning objects indicate their preference for a given action by assigning a vote to that action. The magnitude of the vote corresponds to the intensity of the preference and its sign indicates approval or disapproval. Each reasoning object must assign some vote to every action in the action space. This information is expressed as the *action matrix*. For the experiments reported in this paper, we used the following simple 3×3 action matrix:

decel-left	nil-left	accel-left
decel-nil	nil-nil	accel-nil
decel-right	nil-right	accel-right

If smoother control is desired, an action matrix with more rows and/or columns may be used.

Since different reasoning objects can return different recommendations for the same action, conflicts must be resolved and a good action selected. SAPIENT uses a voting arbiter to perform this assimilation. During arbitration, the votes in each reasoning object's action matrix are multiplied by a scalar weight, and the resulting matrices are summed together to create the *cumulative action matrix*. The action with the highest cumulative vote is selected for execution in the next time-step. This action is sent to the controller and converted into actuator commands (steering and velocity control).

3.3. Parameters

As described in Section 3.1, different reasoning objects use different internal algorithms. Each reasoning object's evaluation depends on a variety of *internal parameters* (e.g. thresholds, gains etc) and the resulting action matrices are then scaled by weights, known here as *external parameters*. Since this parameter space grows linearly with the number of reasoning objects in the SAPIENT module, tuning these parameters manually becomes a time-consuming and error-prone task.

When a new reasoning object is being implemented, it is difficult to determine whether poor performance should be attributed to a bad choice of parameters, a bug within the new module or, more seriously, to a poor representation scheme (inadequate configuration of reasoning objects). To overcome this difficulty, we implemented a method for automatically configuring the parameter space. A total of twenty parameters, both internal and external, were selected for this test, and each such parameter was discretized into eight values (to be represented as a three-bit string). For internal parameters, whose values are expected to remain within a certain small range, we selected a linear mapping (where the three bit string represented integers from 0 to 7); for the external parameters (weights), we used an exponential representation (with the three

```
****** Initialize Probability Vector ******
for i := 1 to LENGTH do P[i] = 0.5;

while (NOT termination condition)
      ****** Generate Samples ******
      for i := 1 to SAMPLES do
            sample_vectors[i] := generate_sample_vector_according_to_probabilities(P);
            evaluations[i] := Evaluate_Solution( sample_vectors[i]; );
      best_vector := find_vector_with_best_evaluation( sample_vectors, evaluations );

      ****** Update Probability Vector towards best solution ******
      for i := 1 to LENGTH do
            P[i] := P[i] * (1.0 − LR) + best_vector[i] * (LR);

      ****** Mutate Probability Vector ******
      for i := 1 to LENGTH do
            if (random (0,1) < MUT_PROBABILITY) then
                  if (random (0,1) > 0.5) then mutate_direction := 1;
                  else mutate_direction := 0;
                  P[i] := P[i] * (1.0 − MUT_SHIFT) + mutate_direction * (MUT_SHIFT);


USER DEFINED CONSTANTS (Values Used in this Study):
SAMPLES: the number of vectors generated before update of the probability vector (100)
LR: the learning rate, how fast to exploit the search performed (0.1).
LENGTH: the number of bits in a generated vector (3 * 20)
MUT_PROBABILITY: the probability of a mutation occuring in each position (0.02).
MUT_SHIFT: the amount a mutation alters the value in the bit position (0.05).
```

Figure 4: PBIL algorithm, explicit preservation of best solution from one generation to next is not shown.

bit string mapping to weights of 0 to 128). The latter representation increases the range of possible weights at the cost of sacrificing resolution at the higher magnitudes. Using a longer bit string would allow finer tuning but increase learning time. In the next section, we describe the evolutionary algorithm used for the learning task.

4. Population Based Incremental Learning

Population-based incremental learning (PBIL) is a combination of evolutionary optimization and hill-climbing [2, 1]. The object of the algorithm is to create a real valued probability vector which, when sampled, reveals high quality solution vectors with a high probability. For example, if a good solution to a problem can be encoded as the string "0101…", a suitable final probability vector would be $(0.01, 0.99, 0.01, 0.99, \ldots)$.

Initially, the values of the probability vector are set to 0.5. Sampling from this vector yields random solution vectors because the probability of generating a 0 or a 1 is equal. As search progresses, the values in the probability vector gradually shift to represent high evaluation solution vectors. This is accomplished as follows: A number of solution vectors are generated based upon the probabilities specified in the probability vector. The probability vector is then pushed towards the best solution vector from set. The distance the probability vector is pushed depends upon the learning rate parameter. After the probability vector is updated, a new set of solution vectors is produced by sampling from the updated vector, and the cycle is continued. Each update of the probability vector marks the end of a *generation*. As the search progresses, entries in the probability vector move away from their initial settings of 0.5 towards either 0.0 or 1.0. The probability vector can be viewed as a prototype vector for generating solution vectors which have high evaluations with respect to the available knowledge of the search space. The full algorithm is shown in Figure 4.

Because of the small size of the population used, and the probabilistic generation of solution vectors, it is possible that a good vector may not be created in every generation. Therefore, in order to avoid moving towards unproductive areas of the search space, the best vector from the previous population is included in the current population (by replacing the worst member of the current population). This solution vector is only used if the current generation does not produce a better solution vector. In genetic algorithm literature, this technique of preserving the best solution vector from one generation to the next is termed *elitist selection*, and is used to prevent the loss of good solutions once they are found.

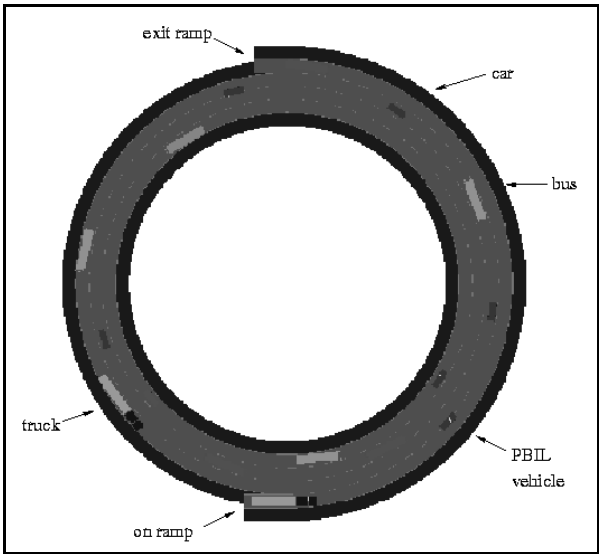

Figure 5: The *cyclotron* test track

Our application challenges PBIL in a number of ways. First, since a vehicle's decisions depend on the behavior of other vehicles which are not under its control, each simulation can produce a different evaluation for the same bit string. We must evaluate each set of vehicle parameters multiple times to compensate for the stochastic nature of the environment. Second, the PBIL algorithm is never exposed to all possible traffic situations (thus making it impossible to estimate the "true" performance of a PBIL string). Third, since each evaluation takes considerable time to simulate, it is important to minimize the number of evaluations needed to learn a good set of parameters.

5. Scenario

All of the tests described below were performed on the track shown in Figure 5, known as the *Cyclotron*. While this highway configuration is not encountered in real-life, it has several advantages as a testbed:

1. It is topologically identical to a highway with equally spaced exits.
2. The entire track can be displayed on a workstation screen.
3. One can create challenging traffic interactions at the entry and exit merges with only a small number of vehicles.
4. Taking the nth exit is equivalent to traveling n laps of the course.

Each scenario was initialized with one PBIL vehicle, and eight rule-based cars (with hand-crafted decision trees). The PBIL car was directed to take the second exit (1.5 revolutions) while the other cars had goals of zero to five laps. Whenever the total number of vehicles on the track dropped below nine, a new vehicle was injected at the entry ramp (with the restriction that there was always exactly one PBIL vehicle on the course).

Whenever a PBIL vehicle left the scenario (upon taking an exit, or crashing), its evaluation was computed based on statistics collected during its run. This score was used by the PBIL algorithm to update the probability vector — thus creating better PBIL vehicles in the next generation.

While driving performance is often subjective, all good drivers should display the following characteristics: they should drive without colliding with other cars, try to take the correct exit, maintain their desired

velocity and drive without straddling the lane markers. Additionally, they should always recommend some course of action, even in hopeless situations.

We encoded the above heuristics as an evaluation function:

$$\begin{aligned} \text{Eval} \quad = \quad & (-10000 \times \text{all-veto})(-1000 \times \text{num-crashes})(-500 \times \text{if-wrong-exit}) \\ & (-0.02 \times \text{accum-speed-deviation})(-0.02 \times \text{accum-lane-deviation})(+1.0 \times \text{dist-traveled}) \end{aligned}$$

where:

- `all-veto` indicates that the PBIL vehicle has extreme objections to all possible actions. With good parameters, this should only occur just before a serious crash.
- `num-crashes` is the number of collisions involving the PBIL vehicle.
- `if-wrong-exit` is a flag, which is true if and only if the PBIL vehicle exited prematurely, or otherwise missed its designated exit.
- `accum-speed-deviation` is the difference between desired and actual velocities, integrated over the entire run.
- `accum-lane-deviation` is the deviation from the center of a lane, integrated over the entire run.
- `dist-traveled` is the length of the run, in meters; this incremental reward for partial completion helps learning.

While the evaluation function is a reasonable measure of performance, it is important to note that there may be cases when a "good" driver is involved in unavoidable crashes; conversely, favorable circumstances may enable "bad" vehicles to score well on an easy scenario. To minimize the effects of such cases, we tested each PBIL string in the population on a set of four scenarios. In addition to normal traffic, these scenarios also included some pathological cases with broken-down vehicles obstructing one or more lanes.

6. Results

We performed a series of experiments using a variety of population sizes, evaluation functions and initial conditions. The evaluation of vehicles using the learned parameters in each case were found to be consistent. This indicates that our algorithms are tolerant of small changes in evaluation function and environmental conditions, and that PBIL is reliably able to optimize parameter sets in this domain. Figure 6 shows the results of one such evolutionary experiment with a population size of 100. This 3-D histogram displays the distribution of vehicles scoring a certain evaluation for each generation. It is clear that as the parameters evolve in successive generations, the average performance of vehicles increases and the variance of evaluations within a generation decreases. Good performance of some vehicles in the population is achieved early (by generation 5) although consistently good evaluations are not observed until generation 15. The number of vehicles scoring poor evaluations drops rapidly until generation 10, after which there are only occasional low scores. The PBIL strings converge to a stable set of parameters and by the last generation, the majority of the PBIL vehicles are able to circle the track, take the proper exit, and avoid crashes.

It should be noted, however, that even cars created in the final generation are not guaranteed to work perfectly. This is because the parameters are generated by sampling the probability vector. Therefore, it is possible, though unlikely in later generations, to create cars with bad sets of parameters. Furthermore, not all accidents are avoidable: they may be caused by dangerous maneuvers made by the other vehicles in response to the difficult traffic situations that often arise in tactical driving domains.

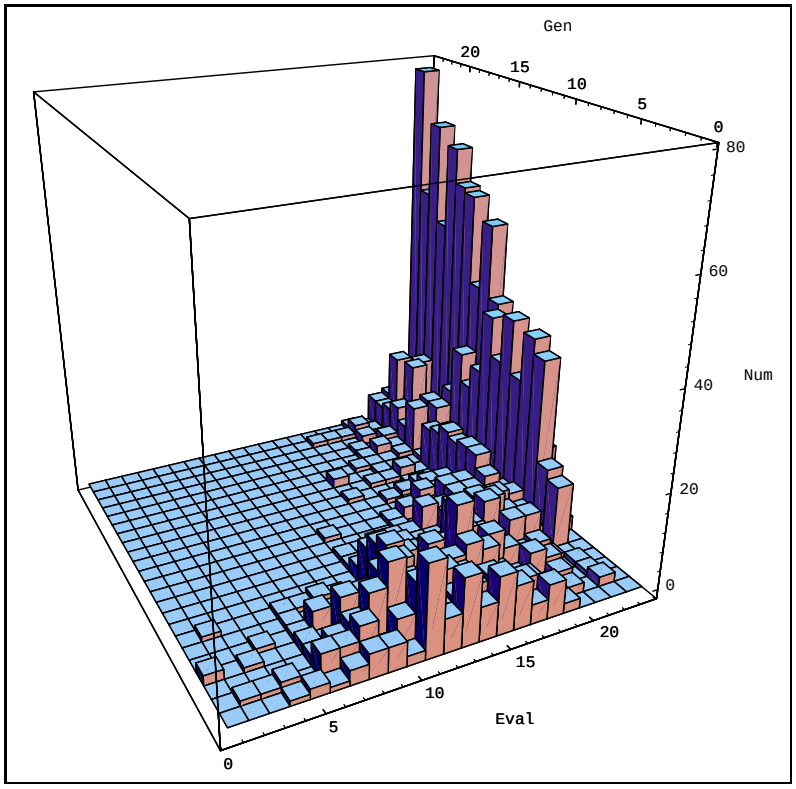

Figure 6: 3-D Histogram showing increase of high-scoring PBIL strings over successive generations. Population size is 100 cars in each generation.

7. Conclusion

Our experiments demonstrate the potential for intelligent behavior in the driving domain using a set of distributed, adaptive, reasoning agents. The SAPIENT architecture allows vehicles to appropriately react to changing driving conditions by activating specific reasoning objects. Using PBIL, we are able to painlessly (for us, not the computer) prototype new reasoning objects in this architecture, and determine whether the reasoning object configurations are suitable for the given traffic scenarios.

In the near future, we plan to extend the reasoning object paradigm to include aggregates of reasoning objects (to allow integration of non-local knowledge). We also hope to explore the possibility of learning altruistic behavior in a collection of PBIL vehicles optimizing a shared evaluation function. We are also developing reasoning objects to tackle additional complications such as vehicle dynamics and noisy sensors.

References

[1] S. Baluja. An empirical comparison of seven iterative and evolutionary function optimization strategies. Technical Report CMU-CS-95-193, Carnegie Mellon University, 1995. To appear in: *The International Conference on Systems Engineering*. Full report available via anonymous ftp at: `reports.adm.cs.cmu.edu`.

[2] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In *Proceedings of the International Conference on Machine Learning (ML-95)*, 1995.

[3] S. Baluja, R. Sukthankar, and J. Hancock. Prototyping intelligent vehicle modules using evolutionary algorithms. In D. Dasgupta and Z. Michalewicz, editors, To appear in *Evolutionary Algorithms in*

Engineering Applications. Springer-Verlag, 1996.

[4] J. Cremer, J. Kearney, Y. Papelis, and R. Romano. The software architecture for scenario control in the Iowa driving simulator. In *Proceedings of the 4th Computer Generated Forces and Behavioral Representation*, May 1994.

[5] E. Dickmanns and A. Zapp. A curvature-based scheme for improving road vehicle guidance by computer vision. In *Proceedings of the SPIE Conference on Mobile Robots*, 1986.

[6] K. Gardels. Automatic car controls for electronic highways. Technical Report GMR-276, General Motors Research Labs, June 1960.

[7] T. Jochem, D. Pomerleau, B. Kumar, and J. Armstrong. PANS: A portable navigation platform. In *Proceedings of IEEE Intelligent Vehicles*, 1995.

[8] T. Jochem, D. Pomerleau, and C. Thorpe. Vision guided lane transitions. In *Proceedings of IEEE Intelligent Vehicles*, 1995.

[9] I. Masaki, editor. *Vision-Based Vehicle Guidance*. Springer-Verlag, 1992.

[10] J. Michon. A critical view of driver behavior models: What do we know, what should we do? In L. Evans and R. Schwing, editors, *Human Behavior and Traffic Safety*. Plenum, 1985.

[11] D. Pomerleau. *Neural Network Perception for Mobile Robot Guidance.* PhD thesis, Carnegie Mellon University, February 1992.

[12] A. Ram, R. Arkin, G. Boone, and M. Pearce. Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation. *Adaptive Behavior*, 2(3):277–305, 1994.

[13] A. Ram and J. Santamaría. Multistrategy learning in reactive control systems for autonomous robotic navigation. *Informatica*, 17(4):347–369, 1993.

[14] D. Reece. *Selective Perception for Robot Driving*. PhD thesis, Carnegie Mellon University, May 1992.

[15] J. Rillings and R. Betsold. Advanced driver information systems. *IEEE Transactions on Vehicular Technology*, 40(1), February 1991.

[16] R. Sukthankar. RACCOON: A Real-time Autonomous Car Chaser Operating Optimally at Night. In *Proceedings of IEEE Intelligent Vehicles*, 1993.

[17] R. Sukthankar. Situational awareness for driving in traffic. Thesis Proposal, October 1994.

[18] R. Sukthankar, J. Hancock, D. Pomerleau, and C. Thorpe. A simulation and design system for tactical driving algorithms. In *Proceedings of AI, Simulation and Planning in High Autonomy Systems*, 1996.

[19] R. Sukthankar, D. Pomerleau, and C. Thorpe. SHIVA: Simulated highways for intelligent vehicle algorithms. In *Proceedings of IEEE Intelligent Vehicles*, 1995.

[20] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer. Vision and navigation for the Carnegie Mellon NAVLAB. *IEEE Transactions on PAMI*, 10(3), 1988.

[21] R. von Tomkewitsch. Dynamic route guidance and interactive transport management with ALI-Scout. *IEEE Transactions on Vehicular Technology*, 40(1):45–50, February 1991.

[22] R. Wallace, A. Stentz, C. Thorpe, W. Moravec, H. Whittaker, and T. Kanade. First results in robot road-following. In *Proceedings of the IJCAI*, 1985.