Appears in: Evoluationary Algorithms in Engineering Applications,
(Ed: D. Dasgupta and Z. Michalewicz), Springer-Verlag, 1998

Prototyping Intelligent Vehicle Modules Using Evolutionary Algorithms

Shumeet Baluja, Rahul Sukthankar, John Hancock
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
E-mail: {baluja|rahuls|jhancock}@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 an automated way to find good parameter settings.

1. Introduction

The task of driving can be characterized as consisting of three levels: strategic, tactical and operational [13]. 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 (operational) level, these maneuvers are translated into control operations.

Mobile robot research has successfully addressed the three levels to different degrees. Strategic-level planners [17, 23] 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 [7, 12] to

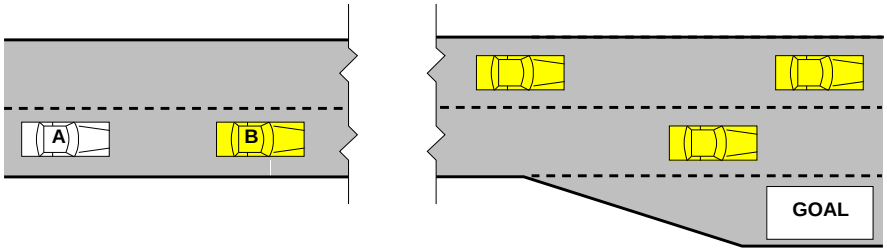

Figure 1: Car A is approaching its desired exit behind a slow vehicle B. Should Car A attempt to pass?

autonomous driving in a variety of situations [5, 18, 14]. Substantial progress in autonomous navigation in simulated domains has also been reported in recent years [16, 4, 15]. 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 [22, 10] 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 real traffic is expensive, risky and potentially disastrous. SHIVA[1] (**S**imulated **H**ighways for **I**ntelligent **V**ehicle **A**lgorithms) [21, 20] 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 ac-

[1] For further information, see <http://www.cs.cmu.edu/~rahuls/shiva.html>

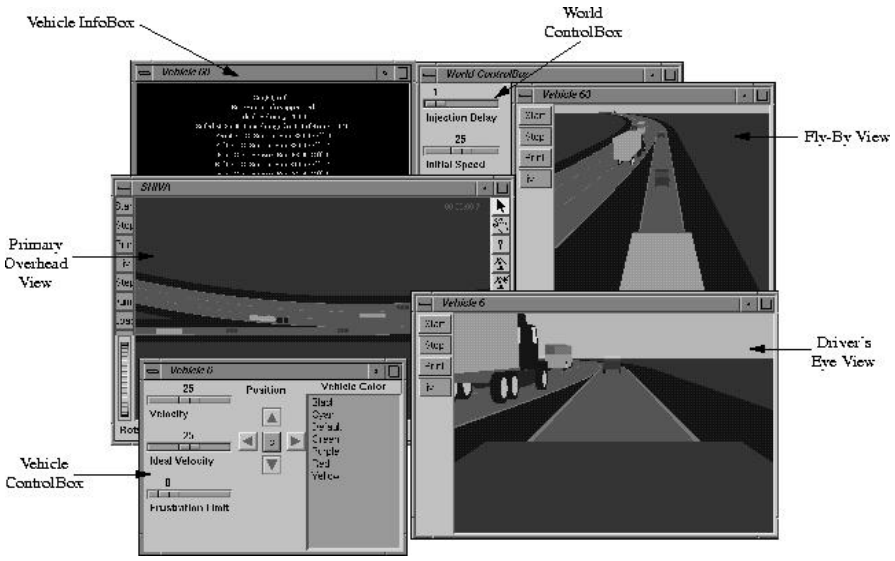

Figure 2: SHIVA: A design and simulation tool for developing intelligent vehicle algorithms.

celerator, 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 comprehensively covered in [21] and details of the design tools may be found in [20].

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).

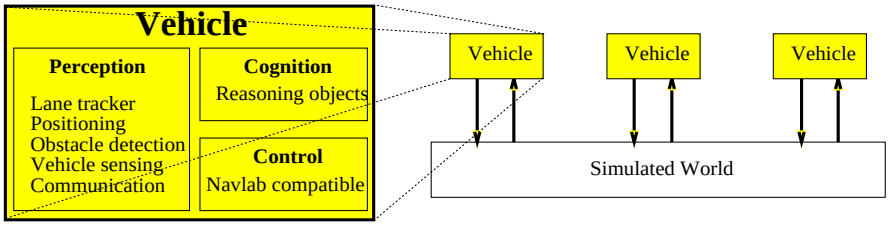

Figure 3: Each vehicle is composed of three subsystems which interact with the simulated world.

2.1. Perception

The perception subsystem consists of a suite of simulated functional sensors (e.g. global positioning systems, 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 [24] 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 [11]. 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 system, which was manually designed, 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. For example, a rule included in car passing maneuvers is:

> "Initiate a left lane change if the vehicle ahead is moving slower than $f(v)$ m/s, and is closer than $h(v)$, and if the lane to your left is marked for legal travel, and if there are no vehicles in that lane within $g(v)$ meters, and if the desired right-exit is further than $e(x, y, v)$ meters."

where: $f(v)$ is the desired car following velocity, $h(v)$ is the desired car following distance (headway), $g(v)$ is the required gap size for entering an adjacent lane, and $e(x, y, v)$ is a distance threshold to the exit based on current lane, distance to exit and velocity. As the maneuver is initiated, the vehicle moves from the *lane tracking*

to the *lane changing* state. While this system performs well on many scenarios, it suffers from four disadvantages: 1) as the example above illustrates, realistic rules require the designer to account for many factors; 2) modification of the rules is difficult since a small change in desired behavior can require many non-local modifications; 3) hand-coded rules perform poorly in unanticipated situations; 4) 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 monolithic state-machine implementation for scenario control. To address some of these problems, we have developed a distributed reasoning architecture, SAPIENT, which is discussed 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 (**S**ituational **A**wareness **P**lanner **I**mplementing **E**ffective **N**avigation in **T**raffic) [19] 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 aspect is represented by an independent module known as a *reasoning object*.

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). In SAPIENT, each reasoning object is responsible for maintaining the relevant state information. The following reasoning objects were used in the scenar-

ios described in this paper:

Reasoning Object	Monitors
Inertia	None: blindly favors constant velocity, same lane.
Velocity	Self state: current and desired velocity
Lane	Self state: lane keeping, lane changing
Exit	Self state: desire to exit; distance to exit
Lead car	Position, velocity and size of car ahead
Front-right car	Position, velocity and size of front-right car
Front-left car	Position, velocity and size of front-left car
Back-right car	Position, velocity and size of back-right car
Back-left car	Position, velocity and size of back-left car

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 over 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 are greatly different, we chose to encode longitudinal motion in terms of changes in velocity, and lateral motion as changes in displacement. Thus the null action represents maintaining speed at the current lane offset.

Tactical maneuvers (such as lane changing) are composed by concatenating several basic actions. Reasoning objects indicate their preference for a basic 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 in the *action matrix*. For the experiments reported in this paper, we used

the following simple 3×3 action matrix:

decelerate-left	nil-left	accelerate-left
decelerate-nil	nil-nil	accelerate-nil
decelerate-right	nil-right	accelerate-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 integration. 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 output depends on a variety of *internal parameters* (e.g. thresholds, gains, etc.). The outputs are then scaled by *external parameters* (e.g. weights).

When a new reasoning object is being implemented, it is difficult to determine whether a vehicle's 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 have implemented a method for automatically configuring the parameter space. A total of twenty parameters, both internal and external, were selected for the tests described here, and each parameter was discretized into eight values (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, we used an exponential representation (with the three 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. A representation with more bits per parameter would allow finer tuning but increase the learning times. Empirically, we found that three bits per parameter allowed good solutions to be rapidly discovered. The encoding is illustrated in Figure 4. In the next section, we describe the evolutionary algorithm used for the learning task.

4. Population-Based Incremental Learning

This section provides a brief introduction to Population-Based Incremental Learning (PBIL), and its relation to genetic algorithms (GAs). PBIL is a combination of

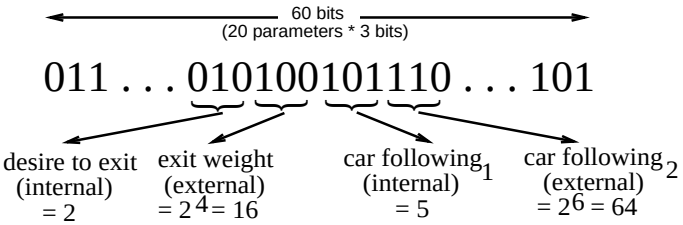

Figure 4: The three-bit encoding scheme used to represent parameters: internal parameters are linearly scaled while external ones are exponentially scaled.

genetic algorithms and competitive learning [1, 3]. The PBIL algorithm attempts to explicitly maintain statistics about the search space and uses them to direct its exploration. The object of the algorithm is to create a real valued probability vector which, when sampled, reveals high quality solution vectors with high probability. For example, if a good solution can be encoded as a string of alternating 0's and 1's, a suitable final probability vector would be 0.01, 0.99, 0.01, 0.99, etc. The full algorithm is shown in Figure 5.

Initially, each element of the probability vector is initialized to 0.5. Sampling from this vector yields random solution vectors because the probability of generating a 0 or 1 is equal. As search progresses, the values in the probability vector gradually shift to represent high evaluation solution vectors through the following process. A number of solution vectors are generated based upon the probabilities specified in the probability vector. The probability vector is pushed towards the generated solution vector with the highest evaluation. After the probability vector is updated, a new set of solution vectors is produced by sampling from the updated probability vector, and the cycle is continued. 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.

One key feature of the early generations of genetic optimization is the parallelism in the search; many diverse points are represented in the population of points during the early generations. When the population is diverse, crossover is an effective means of search, since it provides a method to explore novel solutions by combining different members of the population. Since a GA uses a full population, while PBIL only uses a single probability vector, PBIL may seem to have less expressive power than a GA. A GA can *represent* a large number of points simultaneously; however, a traditional single population GA will not be able to *maintain* a large number of diverse points. Over a number of generations, sampling errors cause the population to converge around a single point. This phenomenon is summarized below:

> "... the theorem [Fundamental Theorem of Genetic Algorithms [8]] assumes an infinitely large population size. In a finite size population,

```
****** 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 5: PBIL algorithm, explicit preservation of best solution from one generation to next is not shown.

> even when there is no selective advantage for either of two competing alternatives . . . the population will converge to one alternative or the other in finite time (De Jong, 1975; Goldberg & Segrest, ICGA-2). This problem of finite populations is so important that geneticists have given it a special name, genetic drift. Stochastic errors tend to accumulate, ultimately causing the population to converge to one alternative or another" [9].

Diversity in the population is crucial for GAs. By maintaining a population of solutions, the GA is able — in theory at least — to maintain samples in many different regions. Crossover is used to merge these different solutions. However, when the population converges, this deprives crossover of the diversity it needs to be an effective search operator. When this happens, crossover begins to behave like a mutation operator that is sensitive to the convergence of the value of each bit [6]. If all individuals in the population converge at some bit position, crossover leaves those bits unaltered. At bit positions where individuals have not converged, crossover will effectively mutate values in those positions. Therefore, crossover creates new individuals that differ from the individuals it combines only at the bit positions where the mated individuals disagree. This is analogous to PBIL which creates new trial solutions that differ mainly in bit positions where prior good performers have disagreed. More details can be found in [3].

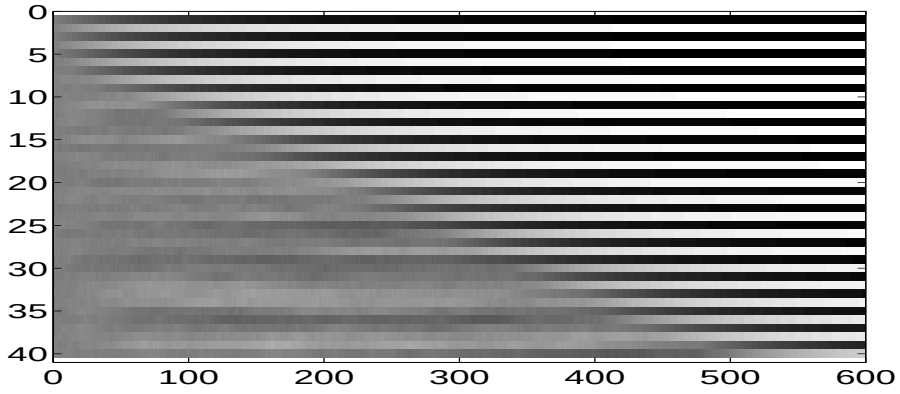

Figure 6: Y axis: bit position in probability vector. X axis: generations. Evolution of the probability vector over successive generations. The value of each element is displayed as a greyscale value. Black and white represent a high probability of generating a 0 and 1 respectively (with grey mapping to intermediate probabilities). Position 0 is the most significant, position 40 is the least. Note that PBIL pins the most significant elements of the probability vector early.

Some of the problems with diversity loss can be addressed by using GAs which employ several independently evolving populations with infrequent interactions. PBIL has also been extended in similar directions with promising results [2]. However, for the tasks explored in this study, these advanced techniques were not needed.

As an example of how the PBIL algorithm works, we can examine the values in the probability vector through multiple generations. Consider the following, simple maximization problem:

$$\text{Eval} = \frac{1.0}{|366503875925.0 - X|}$$

where $0 \leq X < 2^{40}$. Note that 366503875925 is represented in binary as a string of 20 pairs of alternating '01'. The evolution of the probability vector is shown in Figure 6. Note that the most significant bits are pinned to either 0 or 1 very quickly, while the least significant bits are pinned last. This is because during the early portions of the search, the most significant bits yield more information about high evaluation regions of the search space than the least significant bits.

The probabilistic generation of solution vectors does not guarantee the creation of a good solution vector in every iteration. This problem is exacerbated by the small population sizes used in these experiments. 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 GA literature, this technique of preserving the best solution vector from one generation to the next is termed *elitist*

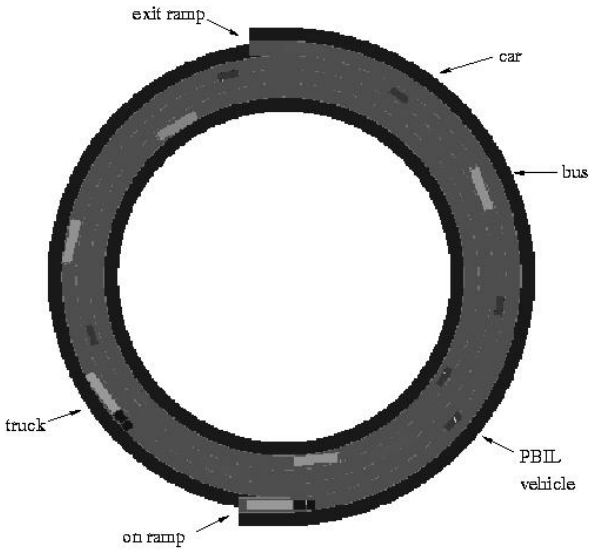

Figure 7: The *cyclotron* test track

selection, and is used to prevent the loss of good solutions once they are found.

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 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, minimizing the total number of evaluations is important.

5. Training Specifics

All of the tests described below were performed on the track shown in Figure 7, 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. Taking the nth exit is equivalent to traveling n laps of the course.
3. One can create challenging traffic interactions at the entry and exit merges with only a small number of vehicles.
4. The entire track can be displayed on a workstation screen.

For training, 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 10 times), 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 at least 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 to be maximized:

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

where:

- `all-veto` indicates that the PBIL vehicle has extreme objections to all possible actions. With good parameters, this should never happen.
- `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 can be cases when a "good" driver becomes 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 light traffic, these scenarios also included some pathological cases with broken-down vehicles obstructing one or more lanes. Adding a wider variety of scenarios,

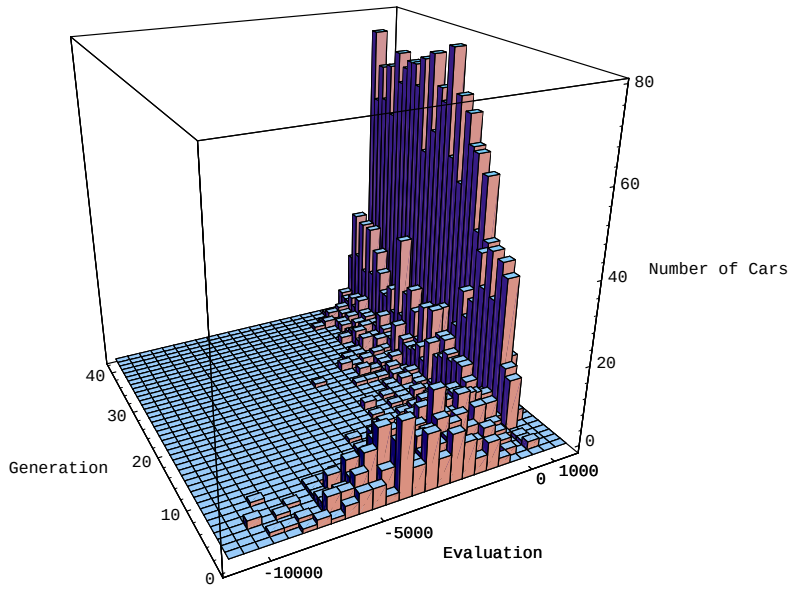

Figure 8: 3-D Histogram showing increase of high-scoring PBIL strings over successive generations. Population size is 100 cars in each generation.

including those in which the traffic congestion levels are higher, has the potential to reveal more comprehensive evaluations of parameter settings. However, the addition of more training scenarios will lead to longer training times.

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 8 shows the results of one such evolutionary experiment with a population size of 100. Also shown are the results from a second experiment with a population size of 20, using the same evaluation function (see Figure 9). For space considerations, only two experiments are shown; however, these experiments have been replicated from a variety of initial starting conditions and parameter settings.

These 3-D histograms display 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. In the experiments with population size 100, good performance of some vehicles in the population is achieved early (by generation 5) although consistently good evaluations are not observed until generation

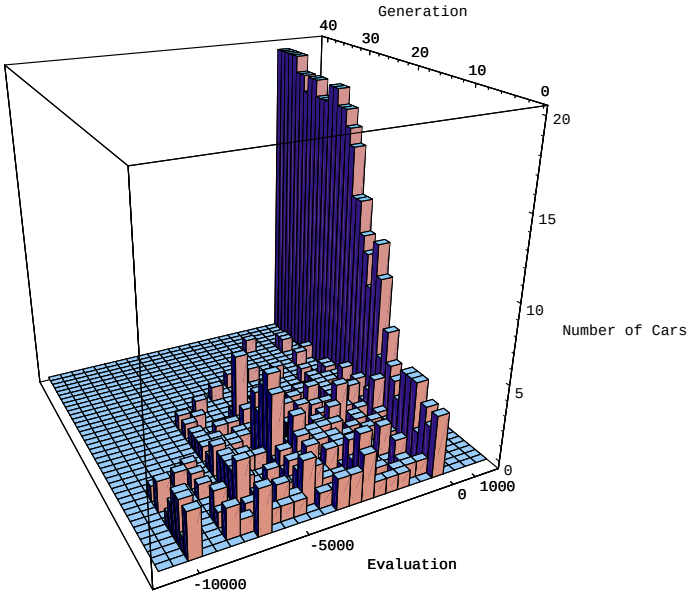

Figure 9: 3-D Histogram showing 20 cars in each generation.

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 in all four scenarios. In the experiments with population size 20, more generations are required before good parameter settings are found; however, fewer evaluations are performed in each generation. Also note that the run with population size 20 does not find as good solutions as that with population size 100. By the end of the run, the majority of evaluations for the size 20 population ranged between 0–500; for the size 100 population, between 1000–1500.

Finally, it should also be noted that even cars created in the final generation are not guaranteed to drive 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.

To visually display the learned behavior of the cars, we created obstacle-avoidance test tracks with numerous stopped cars. The path of a car which successfully navigates around these cars is shown in Figure 10. While all of the stopped cars are avoided, the trajectory followed shows some undesirable oscillations. This is caused by limitations in the reasoning objects used in this test. Since the obstacle avoidance modules were stateless, they did not consistently follow a single course of action.

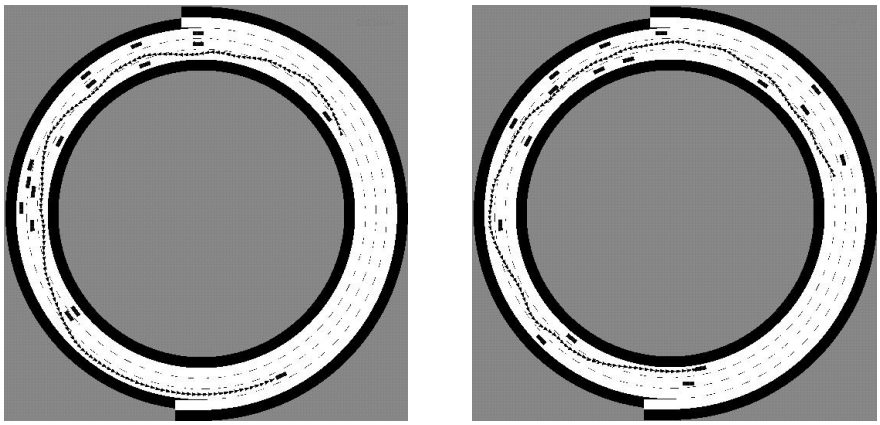

Figure 10: The *Obstacle Avoidance* test tracks. Note that the best vehicles are very aggressive, often avoiding the stopped cars with only narrow margins.

More sophisticated modules which maintain state will overcome these problems.

7. Conclusion and Future Directions

Our experiments have demonstrated: (1) The potential for intelligent behavior in the tactical driving domain using a set of distributed reasoning modules. (2) The ability of evolutionary algorithms to automatically configure a *collection* of these modules for addressing their *combined* task.

In this study, we used a very simple evaluation function. By changing the objective function, we can change the learned behavior of the vehicles. By introducing alternative objective functions, we plan to extend this study in at least three directions. First, since the cars created in this study often perform dangerous maneuvers, we will incorporate penalties for overly aggressive behavior. For example, by introducing penalties for coming too close to another car, the cars should learn to maintain a safety cushion. Second, for automated highways, we would like the cars to exhibit altruistic behavior. In a collection of PBIL vehicles, optimizing a *shared* evaluation function (such as highway throughput) may encourage cooperation. Finally, we are developing reasoning objects to address additional complications which will arise when these vehicles are deployed in the real world, such as complex vehicle dynamics and noisy sensors.

8. Acknowledgements

The authors would like to acknowledge the valuable discussions with Dean Pomerleau and Chuck Thorpe which helped to shape this work. Shumeet Baluja is supported by a graduate student fellowship from the National Aeronautics and Space Administration, administered by the Lyndon B. Johnson Space Center, Houston,

Texas. This work is also partially supported by the Automated Highway System project, under agreement DTFH61-94-X-00001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NASA or the AHS Consortium.

References

[1] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Computer Science, Carnegie Mellon, 1994.

[2] S. Baluja. Genetic algorithms and explicit search statistics. In *Advances in Neural Information Processing Systems 9* (to appear), 1997.

[3] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In A. Prieditis and S. Russell, editors, *Proc. International Conference on Machine Learning (ML-95)*, pages 38–46. Morgan Kaufmann Publishers, 1995.

[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] L. Eshelman. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In Rawlins, editor, *Foundations of Genetic Algorithms*, pages 265–283. Morgan Kaufmann Publishers, 1991.

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

[8] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[9] D. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, San Mateo, CA., 1987. Morgan Kaufmann Publishers.

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

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

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

[13] 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.

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

[15] 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.

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

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

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

[19] R. Sukthankar, J. Hancock, S. Baluja, D. Pomerleau, and C. Thorpe. Adaptive intelligent vehicle modules for tactical driving. In *Proceedings of AAAI workshop on Adaptive Intelligent Agents*, 1996.

[20] 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.

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

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

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

[24] 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.