

Evolving an Intelligent Vehicle for Tactical Reasoning in Traffic

Rahul Sukthankar^{1,2}, Shumeet Baluja^{1,2}, John Hancock²
{rahuls|baluja}@jprc.com, jhancock@ri.cmu.edu

¹Justsystem Pittsburgh Research Center
4616 Henry Street
Pittsburgh, PA 15213

²The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Recent research in automated highway systems has ranged from low-level vision-based controllers to high-level route-guidance software. However there is currently no system for tactical-level reasoning. Such a system should address tasks such as passing cars, making exits on time, and merging into a traffic stream. Our approach to this intermediate-level planning combines a distributed reasoning system (PolySAPIENT) with a novel evolutionary optimization strategy (PBIL). PBIL automatically tunes PolySAPIENT module parameters in simulation by evaluating candidate modules on various traffic scenarios. Since the control interface to the simulated vehicles is identical to that on the Carnegie Mellon Navlab vehicles, modules developed using this process can be directly ported to existing hardware. This method is currently being applied to the automated highway system domain; it also generalizes to many complex robotics tasks where multiple interacting modules must simultaneously be configured without individual module feedback.

1. Introduction

The task of driving can be characterized as consisting of three levels: strategic, tactical and operational [8]. 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 [12] 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 [5, 7] to autonomous driving in a variety of situations [3, 9]. Substantial progress

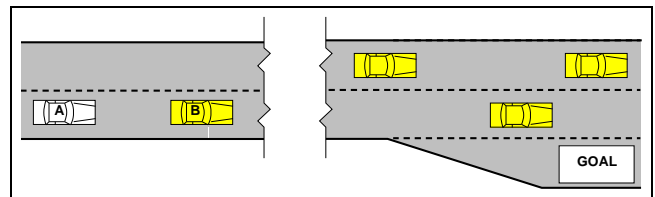


Figure 1: Car A is approaching its desired exit behind a slow vehicle B. Should Car A attempt to pass?

in autonomous navigation in simulated domains has also been reported in recent years [11, 2, 10]. However, the decisions required at the tactical level are difficult and a general solution remains elusive (see Figure 1).

PolySAPIENT, described in Section 3, is a system designed to drive the Carnegie Mellon Navlab [15] in situations similar to the situation shown in Figure 1. PolySAPIENT 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. The SHIVA Simulator

Simulation is essential in developing intelligent vehicle systems because testing new algorithms in real traffic is expensive, risky and potentially disastrous. SHIVA¹ (Simulated Highways for Intelligent Vehicle Algorithms) [14] 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

¹More information and an interactive demo are available at:
<<http://www.cs.cmu.edu/~rahuls/shiva.html>>

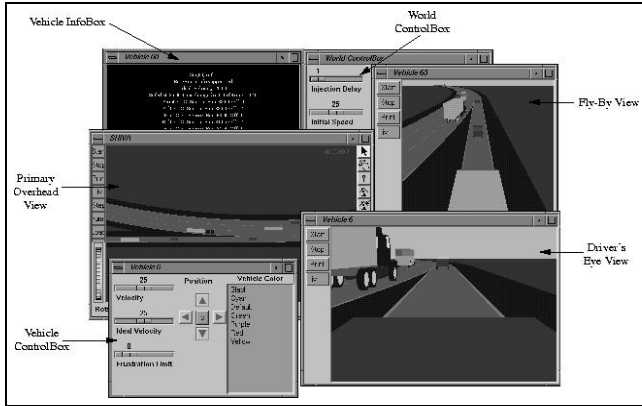


Figure 2: SHIVA: A design and simulation tool for developing intelligent vehicle algorithms.

algorithms direct 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). Details of the simulator are comprehensively covered elsewhere [14, 13]. All vehicles are composed of three subsystems: perception, cognition, and control.

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.

While a variety of cognition modules have been developed in SHIVA, this paper is only concerned with two types: rule-based reasoning (MonoSAPIENT [13]) and a modular, distributed architecture (PolySAPIENT [13]). The rule-based reasoning system, which was manually designed, is implemented as a monolithic decision tree. An example rule 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. 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.* [2] in their single-layer state-machine implementation for scenario control. To address some of these problems, we have developed a distributed reasoning architecture, PolySAPIENT, which is discussed in Section 3.

The control subsystem is compatible with the controller available on the Carnegie Mellon Navlab II robot testbed vehicle. Commands to the controller are issued by the cognition modules at a rate of 10 Hz. SHIVA only allows vehicles to control desired velocity and steering curvature. Denying control over acceleration ensures that systems developed in simulation can be directly ported onto our existing hardware configuration.

3. PolySAPIENT

PolySAPIENT [13] is a distributed reasoning system designed to solve tactical driving problems. To overcome deficiencies with the monolithic reasoning systems, PolySAPIENT partitions the driving task into several aspects; each is represented by an independent module known as a *reasoning object*.

3.1. Reasoning Objects

The reasoning associated with each relevant physical entity in the environment (e.g. the car ahead or the approaching exit), is addressed by a specialized local expert known as a *reasoning object*. Each reasoning object is responsible for assessing the impact of its associated entity on the robot's actions, *independently*, of the interactions with other entities in the scene. Thus, the object associated with monitoring the vehicle ahead is mainly concerned with preventing collisions and suggesting overtaking, while the object associated with the exit recommends lane-changes towards the exit. For this to work, the reasoning objects must share a common output language. In PolySAPIENT, every object presents *votes* and *vetos* over a preselected set of actions (see Section 3.2), and the action selection is performed by a domain-independent arbiter.

A total of nine reasoning objects were used in the experiments described here; these included experts associated with the current lane, the desired exit and nearby vehicles. Appropriate reasoning objects were instantiated or destroyed as the relevant objects appeared and left the simulated sensors' field of view. Additional inputs included higher-level goals (such as the preferred velocity). The rea-

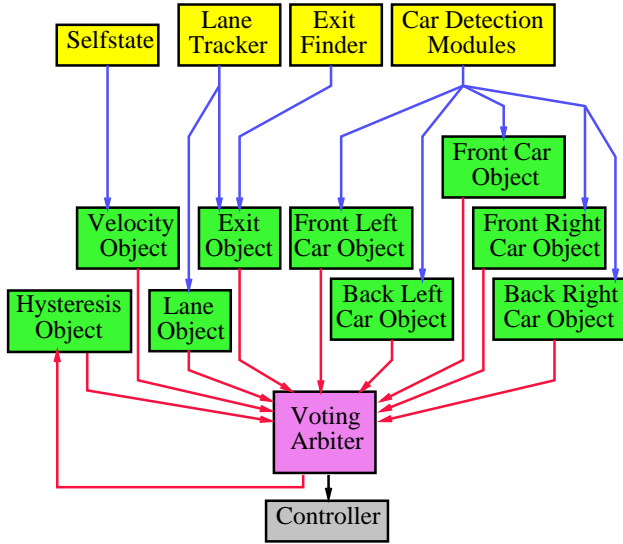


Figure 3: The PolySAPIENT reasoning object configuration consists of several, independent local experts, known as reasoning objects.

soning object configuration is shown in Figure 3 For more details on this architecture, see [13].

3.2. Actions

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. All actions have a velocity change (longitudinal) and a lane-offset (lateral) component; for example, “brake hard while changing left” or “increase speed and maintain your current lane position”. In the experiments described here, there were nine possible actions: the cross product of {increase speed, maintain speed, decrease speed} with {shift-left, maintain lane, shift-right}.

Since different reasoning objects can return different recommendations for the next action, conflicts must be resolved. PolySAPIENT uses a voting arbiter to perform this integration. During arbitration, all of the votes for a given action are summed together (after being scaled by the reasoning object’s external parameter), and the most popular action is executed.

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.

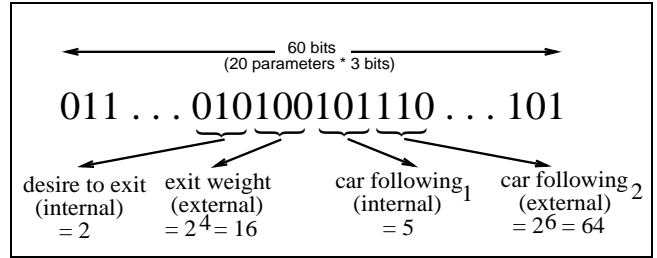


Figure 4: The three-bit encoding scheme used to represent parameters: internal parameters are linearly scaled while external ones are exponentially scaled.

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

Population-Based Incremental Learning (PBIL) is a combination of genetic algorithms (GAs) [6] and competitive learning [1]. 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 ini-

```

***** 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 occurring 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.

tialized 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.

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) — in GA literature, this is termed *elitist selection* [6].

Since space limitations preclude a complete discussion about the relationship between GAs and PBIL, we can only provide a brief intuition. 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 [4]. If all individuals in the population con-

verge 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 [1].

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 6, known as the *Cyclotron*. While this highway configuration is not encountered in real-life, it has several benefits as a testbed: 1) It is topologically identical to a highway with equally spaced exits; 2) Taking the n th 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. 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:

$$\text{Eval} = -(10000 \times \text{all-veto}) - (1000 \times \text{num-crashes})$$

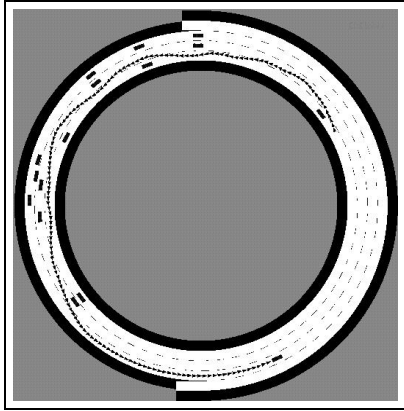


Figure 6: Cyclotron track (15 obstacles)

$$-(500 \times \text{if-wrong-exit}) - (0.02 \times \text{speed-deviation}) \\ -(0.02 \times \text{lane-deviation}) + (\text{dist-traveled})$$

where: `all-veto` indicates that the PBIL vehicle objects to all 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 — true if and only if the PBIL vehicle exited prematurely, or otherwise missed its designated exit; `speed-deviation` is the difference between desired and actual velocities, integrated over the entire run; `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 (an incremental reward for partial completion). The coefficients used in the evaluation function were set by hand. It should be noted that varying these coefficients, even by an order of magnitude, revealed similar results, in terms of missed exits, collisions, speed deviation and lateral handling (See [13] for details).

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. These scenarios varied in traffic density and also included some pathological cases such as 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

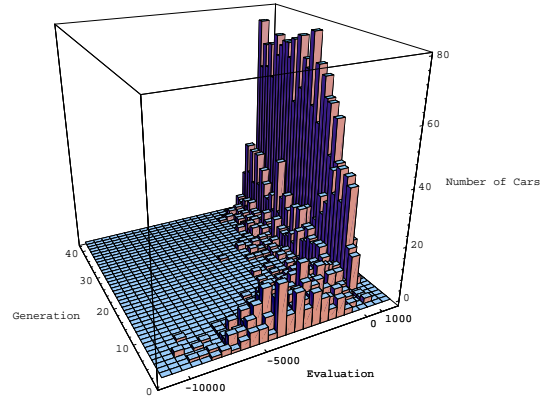


Figure 7: 3-D Histogram showing increase of high-scoring PBIL strings over successive generations. Population size is 100 cars in each generation.

is reliably able to optimize parameter sets in this domain. Figure 7 shows the results of one such evolutionary experiment with a population size of 100. For more experiments, which have been conducted with a variety of initial starting conditions and evaluation functions, see [13].

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 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. Figures 8 and 9 show the progress of the vehicles in greater detail.

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 introducing alternative objective functions, we plan to extend this study in at least two directions. First, 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 through-

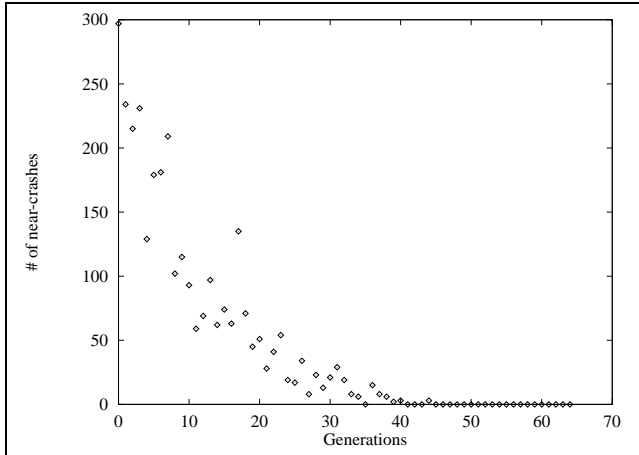


Figure 8: Number of crashes summed over all vehicles in a population, as a function of generation. Note that all vehicles in the later generations avoid crashes.

put) may encourage cooperation. Second, 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. Thanks also to Gita Sukthankar, for the data processing scripts. This work was partially supported by the Automated Highway System project, under agreement DTFH61-94-X-00001.

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, Carnegie Mellon University, 1994.
- [2] 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*, 1994.
- [3] 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.
- [4] L. Eshelman. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In *Foundations of Genetic Algorithms*, pages 265–283. Morgan Kaufmann Publishers, 1991.
- [5] K. Gardels. Automatic car controls for electronic highways. Technical Report GMR-276, General Motors Research Labs, June 1960.

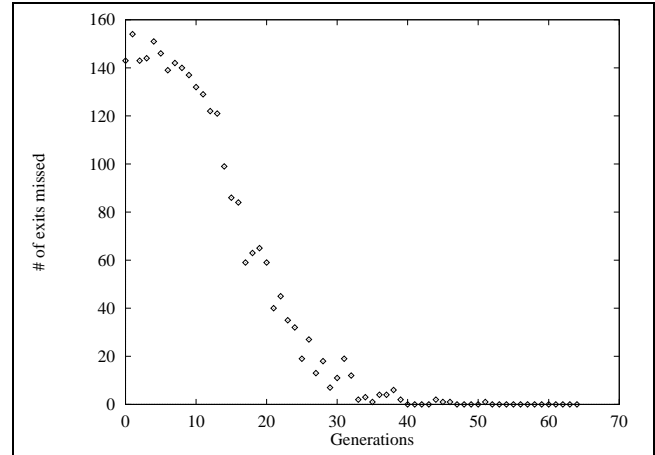


Figure 9: Number of missed exits summed over all vehicles in a population, as a function of generation. Note that all vehicles in the later generations make their desired exits.

- [6] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
- [7] I. Masaki, editor. *Vision-Based Vehicle Guidance*. Springer-Verlag, 1992.
- [8] 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.
- [9] D. Pomerleau. *Neural Network Perception for Mobile Robot Guidance*. PhD thesis, Carnegie Mellon University, February 1992.
- [10] 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), 1994.
- [11] D. Reece. *Selective Perception for Robot Driving*. PhD thesis, Carnegie Mellon University, May 1992.
- [12] J. Rillings and R. Betsold. Advanced driver information systems. *IEEE Transactions on Vehicular Technology*, 40(1), 1991.
- [13] R. Sukthankar. *Situation Awareness for Driving in Traffic*. PhD thesis, Carnegie Mellon University, January 1997.
- [14] R. Sukthankar, J. Hancock, and C. Thorpe. Tactical-level simulation for intelligent transportation systems. To appear in *Journal on Mathematical and Computer Modeling*, 1997. Special Issue on ITS.
- [15] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer. Vision and navigation for the Carnegie Mellon Navlab. *IEEE Transactions on PAMI*, 10(3), 1988.