

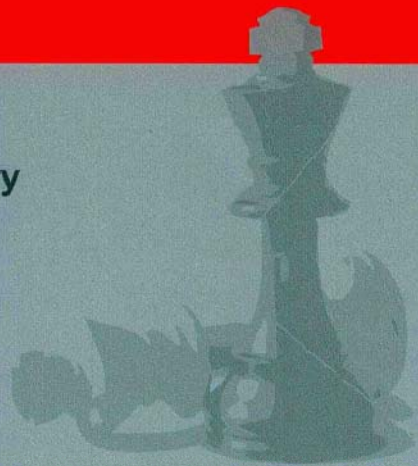
BIRKHÄUSER

Annals of the
International Society
of Dynamic Games

Advances in Dynamic Games and Their Applications

Analytical and Numerical Developments

Pierre Bernhard
Vladimir Gaitsgory
Odile Pourtallier
Editors



Numerical Solution of the Game of Two Cars with a Neurosimulator and Grid Computing

Hans-Jörg von Mettenheim
Institut für Wirtschaftsinformatik
Gottfried Wilhelm Leibniz Universität Hannover
Königsworther Platz 1, D-30167 Hannover, Germany
mettenheim@iwi.uni-hannover.de

Michael H. Breitner
Institut für Wirtschaftsinformatik
Gottfried Wilhelm Leibniz Universität Hannover
Königsworther Platz 1, D-30167 Hannover, Germany
<http://www.iwi.uni-hannover.de>
breitner@iwi.uni-hannover.de

Abstract

The famous game of two cars is a pursuit-evasion dynamic game. In the extended version presented here, a correct driver (evader) on a freeway detects a wrong-way driver (pursuer in a worst case scenario), i.e., a car driving on the wrong lanes of the road or in the wrong direction. The correct driver must try to avoid collision against all possible maneuvers of the wrong-way driver. Additionally, he must try to stay on the freeway lanes. Analytically, the game is not fully solvable. The state-space is cut by various singular manifolds, e.g., barriers, universal, and dispersal manifolds. Here, discretized Stackelberg games are solved numerically for many positions in the state-space. The resulting trajectories and their adherent information are used to synthesize optimal strategies with artificial neural networks. These networks learn the optimal turn rates and optimal velocity change rates. The networks are trained with the high-end neurosimulator FAUN (Fast Approximation with Universal Neural Networks). A grid computing implementation is used which allows significantly shorter computing times. This implementation runs on low-budget, idle PC clusters and moreover power saving allows to wake up and shut down computers automatically. Parallelization on cheap hardware is one of the key benefits of the presented approach as it leads to fast but nonetheless good results. The computed artificial neural networks approximate the Stackelberg strategies accurately. The approach presented here is applicable to many other complex dynamic games which are not (fully) solvable analytically.

Key words. Dynamic game, Stackelberg game, synthesis of optimal strategies, artificial neural networks, parallel computation, grid computing, game of two cars.

AMS Subject Classifications. Primary 91A25; Secondary 68T10, 68M14.

1 The Game of Two Cars Revisited

Even today, several thousand cases of people driving on the wrong side of the road are officially registered on German freeways every year. The actual number is estimated to be several times higher. Often, people are injured or even killed. A collision avoidance device can help to minimize the dangers resulting from wrong-way drivers. This can be modeled as a pursuit-evasion dynamic game. In the present paper, an enhanced version of the game of two cars is investigated. The idea is that the maneuvers of wrong-way drivers are generally unpredictable, e.g., because of drunkenness, fatigue, or panic. It is therefore suitable for the correct driver to assume the worst and to act, as if the wrong-way driver was trying to capture him. The correct driver chooses the optimal strategy against all possible decisions of the wrong-way driver. The resulting dynamic game is modeled as Stackelberg game and the neurosimulator FAUN is used to synthesize strategies.

To simplify our real life problem, we restrict ourselves to a collision avoidance problem for two cars: one is driven by the correct driver E, the other one by the wrong-way driver P. For this collision avoidance problem the kinematic equations can be modeled by:

$$\dot{x}_P = v_P \sin \phi_P, \quad (1)$$

$$\dot{x}_E = v_E \sin \phi_E, \quad (2)$$

$$\dot{y} = v_P \cos \phi_P - v_E \cos \phi_E, \quad (3)$$

$$\dot{v}_E = b_E \eta_E, \quad (4)$$

$$\dot{\phi}_P = w_P u_P, \quad (5)$$

$$\dot{\phi}_E = w_E(v_E) u_E; \quad (6)$$

see Fig. 1. The subscripts P and E of the notations refer to the wrong-way driver (the pursuer) P and the correct driver (the evader) E, respectively. The independent variable t denotes time, the state variables x_P, x_E denote the distance of P and E from the left-hand side of the freeway, and y denotes the distance between P and E orthogonal to the x -direction. The state variables ϕ_P, ϕ_E denote the driving directions of P and E and v_P, v_E are the velocities of P and E. The control variables u_P, u_E denote the turn rates of P and E, and η_E the velocity change rate. Without oversimplification, v_P is taken as constant and the maximum angular velocities $w_P(v_P)$ and $w_E(v_E)$ are prescribed depending on the type of car. The kinematic constraints of bounded radii of curvature are taken into account by the control variable inequality constraints

$$-1 \leq u_P \leq +1, \quad (7)$$

$$-1 \leq u_E \leq +1. \quad (8)$$

Steering $u_P = \pm 1$ and $u_C = \pm 1$ means executing an extreme right/left turn for P and E, respectively. The kinematic constraint of acceleration and deceleration for

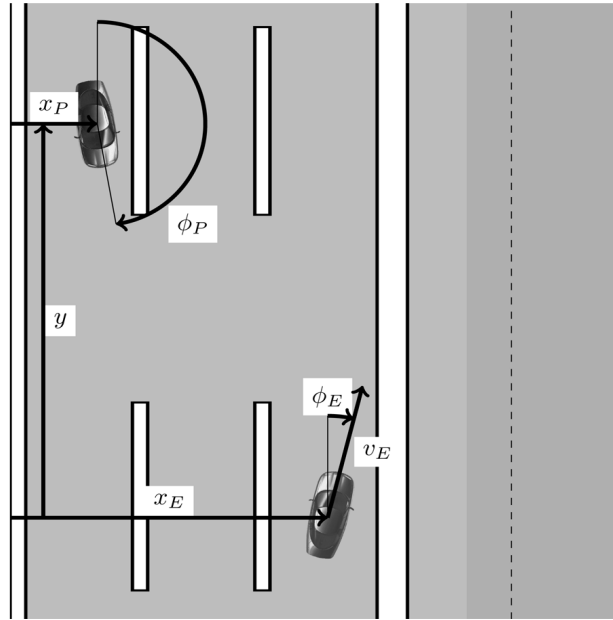


Figure 1: State variables of the collision avoidance problem for two cars. The evader is allowed to be on the edge strip with two wheels. The corresponding state variables for evader and pursuer are denoted with subscripts E and P , respectively.

E is taken into account by the control variable inequality constraint

$$-1 \leq \eta_E \leq +0.48. \quad (9)$$

The constraint

$$0\text{m} \leq x_E \leq 15\text{m} \quad (10)$$

considers the fact that E wants to stay on the freeway. A similar constraint for P is explicitly rejected, as a worst-case situation for E is analyzed. It is indeed imaginable that the wrong-way driver P doesn't care if his car is damaged or he is injured when leaving the freeway.

The maximum deceleration rate is given by $b_E = 10.72\text{m/s}^2$. This corresponds to a modern car. The maximum angular velocities are given by:

$$w_P = \frac{\mu_0 g}{v_P}, \quad (11)$$

$$w_E = \frac{\mu_0 g}{v_E}. \quad (12)$$

The constant $\mu_0 = 0.55$ corresponds to the adhesion coefficient on dry asphalt and $g = 9.81\text{m/s}^2$ is the acceleration of gravity.

The goal of the correct driver E is to avoid collision against all possible maneuvers of P. It can be formulated as

$$\min_{\gamma_P \in \Gamma_P} \max_{\gamma_E \in \Gamma_E} \min_{t_0 \leq t < \infty} d(z(t)) \quad (13)$$

with the vector $z := (x_P, x_E, y, v_E, \phi_P, \phi_E)$ of state variables, with (feedback) strategies $\gamma_P(z)$ and $\gamma_E(z)$, with sets Γ_P and Γ_E of all admissible strategies, and with Euclidean distance $d(z)$ between P and E. The control variables are determined by $u_P(z) := \gamma_P(z)$ and $(u_E(z), \eta_E(z)) := \gamma_E(z)$. The strategy γ_P is admissible, if the constraint (7) is fulfilled for all $t \in [t_0, t_f]$, and the strategy γ_E is admissible, if the constraints (8), (9), and (10) are fulfilled for all $t \in [t_0, t_f]$. The collision avoidance maneuver starts at the time t_0 at $z(t_0) = z_0$, when the correct driver E notices P. The collision avoidance maneuver ends at the terminal time $t_f < \infty$ when the minimum distance between P and E is reached. Theoretically, it is imaginable that the game continues if P turns on the street. This case is deliberately ignored.

The present collision avoidance game is an enhancement of the game of two cars as introduced, e.g., in [23]. It can be embedded in the theory of pursuit and evasion. If the state constraint (10) is omitted and constant velocity v_E is assumed, the present game corresponds to the game of two cars. If additionally the correct driver E needs not obey the constraint of bounded curvature, i.e., $u_E \in]-\infty, \infty[$, the homicidal chauffeur game arises. In [33] a neural network solution for this relatively simple game is presented. Nevertheless, the full solutions of the game of two cars and even the homicidal chauffeur game are extraordinarily baffling. The state-space is cut by diverse singular manifolds. Neither the game of two cars nor the homicidal chauffeur game can be solved fully, i.e., optimal strategies $\gamma_P^*(z)$ and $\gamma_E^*(z)$ cannot be calculated explicitly for all z , for all velocities v_P and v_E , and for all maximum angular velocities w_P and w_E . Nevertheless various collision avoidance problems have been investigated; see, e.g., [3, 4, 13–15, 26–30, 34, 37–39, 42, 44, 46, 47].

As the goal of this paper is not to give an analytical solution for the presented enhanced game of two cars, but to show that even complex dynamic games can be solved with neural networks, a more basic approach is chosen; see also [7]. Therefore, the computation of strategies is realized with a Stackelberg game. First, E optimizes his strategy against all possible maneuvers of P. Then, P optimizes his strategy against the well-known strategy of E. This very conservative assumption is unfair for E but is part of our worst-case analysis.

It has to be noted, that the need to compute a solution first, albeit numerical, presents a potential limitation. The required numerical computation is time consuming and can only be difficultly realized in real-time. This is, of course, one of the reasons, why artificial neural networks are used.

2 The Neurosimulator FAUN

The following excursus explains the basics of a neurosimulator and the advantages of parallelization. Generally, a neurosimulator can be useful when collected data has to be connected even if classical methods like linear or nonlinear regression don't work. At the Institut für Wirtschaftsinformatik in Hanover, the neurosimulator FAUN (Fast Approximation with Universal Neural Networks) has been under active development since 1996.

The process of finding suitable artificial neural networks is called training. This training can take CPU hours or even days on a single personal computer. This is explained by the fact that several networks are tried, normally between 100 and 10000. The trained neural networks have no mutual dependencies. It is possible to split the training task on several threads. This is meant by coarse-grained parallelization in contrast to fine grained parallelization. The advantage of coarse-grained parallelization is that the communication network can be quite slow, standard fast Ethernet with 100 Mbit/s is sufficient. The parallel version of FAUN runs on homogeneous computer clusters as well as on heterogeneous personal computer networks using a parallel programming package which has to be previously installed and configured. This is a disadvantage and inspired the authors to the development of the grid computing client briefly described in this paper.

The Hanover School of Economics owns a student and staff computer cluster with — amongst others — 38 Pentium IV computers at 2.66 GHz running the operating system Windows XP professional. As the cluster is rarely used and almost only for special lectures the combined power of these computers is free most of the time. This leads to the idea of using the unused capacities for calculations and therefore reducing the need of an external computation-only cluster or an expensive supercomputer. The condition is that other occasional users are not disturbed and that the configuration necessities are low. To make the configuration more ecological, a mechanism is needed that allows to wake up the computers from the standby state and to shut them down again.

The currently available parallel programming packages cannot fulfill all these wishes. They require configuration and wake up or shutdown is not implemented. This means that an alternative concept has to be developed. Our new grid computing client is installed by simply copying the files onto the computer and is further on managed via a web interface. As modularity is a main design goal, the client cannot only cooperate with the neurosimulator but offers functionalities that make it possible to hook up other programs. For the basic programming paradigms used, see [1, 17, 18, 22, 24, 35, 40, 41].

In the following the concepts of low-budget grid computing and the FAUN grid computing client are briefly discussed to give an idea of the software used for the strategy synthesis. Reachable economies at the School of Economics are highlighted. Then, the Stackelberg game used to generate trajectories for the neural network training is presented. Finally, the results of neural network guidance compared to the equivalent Stackelberg game are discussed.

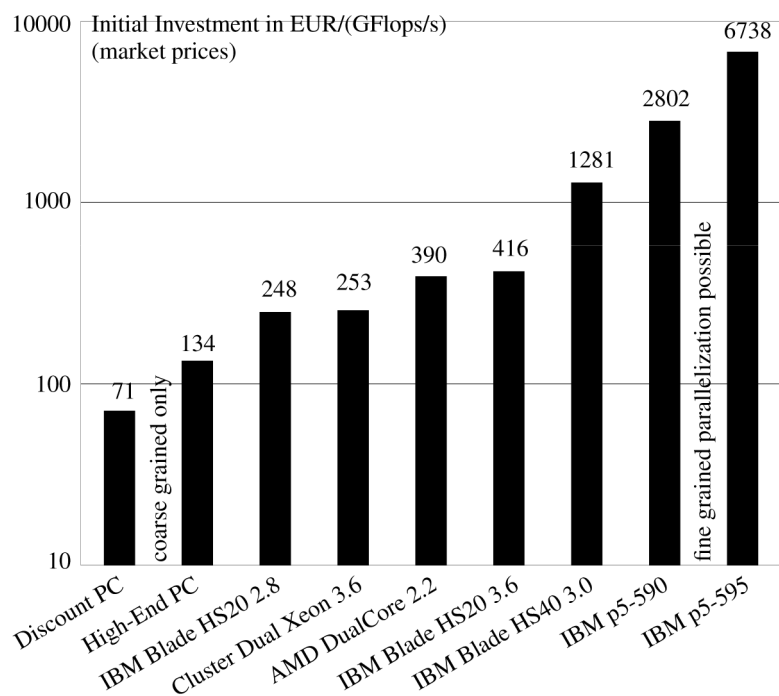


Figure 2: Price comparison of different computer types for coarse-grained parallelization. Note that costly computers offer more advantages than simple CPU speed and that therefore it is not advisable to use a cluster for massively parallel tasks.

3 The FAUN Grid Computing Client

Grid computing is normally used when high performance in a broad sense is needed. This can mean high availability or HPC in its actual sense, i.e., raw computing power is required. It is worth looking at the significant differences in the initial investment for obtaining roughly the same computing power; see Fig. 2. Ranging from affordable 71 Euros per GFlops/s with a standard personal computer the range goes up to more than 6000 Euros per GFlops/s with a massively parallel IBM supercomputer. Yet, these two extremes are not identical. The standard personal computer comes without enhanced manufacturer support, doesn't offer remote administration features and — perhaps most important of all — has a comparatively slow interprocessor communication. Indeed the maximum available speed is 1 GBit/s with a standard network card. On the other hand, the supercomputer ships with 32 CPUs which are mutually connected at a 100times that speed. The 32 CPUs have two cores and correspond to $(2 \times 32 =) 64$ conventional CPUs.

Therefore, Fig. 2 has to be read carefully and the potential customer has to decide, whether high interprocessor communication speeds are important or not so relevant for the given task. In several cases and also with a neurosimulator the answer is clear: interprocessor communication speed isn't the bottleneck. It is advisable to use standard hardware. However, it is not enough to simply buy the hardware. The computers also have to communicate with one another in a sensible way so that the required task is achieved. That is the reason why certain manufacturers offer so-called blades. These are made of mostly standard CPUs but have a very compact design and are easy to manage remotely. Although not offering the advantages of massively parallel systems they cost at least approximately four times as much as a standard computer even for a small blade. Most institutions have unused computing capacities like clusters or staff computers. This leads to the wish to use these resources comfortably. A middleware is needed that interfaces the actual computational application, e.g., the neurosimulator, with basic management functions being file transfer, message passing and power management.

The importance of the latter point is shown in Fig. 3. Actually, the cluster of the School of Economics for students and staff with its 38 Pentium IV computers is powered on only on the five working days from 10 am to 4 pm, i.e., 6 hours per day on roughly 40 weeks per year. The average power consumption is shown in Tab. 3. The standby state means that the computers can be woken up with a so-called magic packet.

It is assumed that the computers are powered on, but mostly without load as is the case when doing simple typewriting or Internet surfing. Using a standard rate of 18.13 ct/kWh the energy costs per year are calculated. The second considered case (full on) happens when the cluster is used for computation *without* power management, i.e., the cluster would be used to full capacity four hours per working day on 40 weeks for computation additionally. If the cluster wasn't powered off during the computation pauses, 3134 Euros would be wasted in idle time every year. The same case *with* power management (upper graph in the figure) would reduce the energy costs considerably.

Another concept — distributed computing — can also provide additional computing capacity nearly for free. Distributed computing describes the fact that the computation is done at various locations and on various types of computers. This can be, e.g., staff computers or laptops all over the university or even at home. The computers only need to be at least temporarily connected via a network. This network is normally an Ethernet or the Internet. The challenge for distributed com-

Table 1: Measured average power consumption (Pentium IV, 2.66 GHz)

state	consumption in Wh
standby	3.4
idle on	67.5
full load	113.3

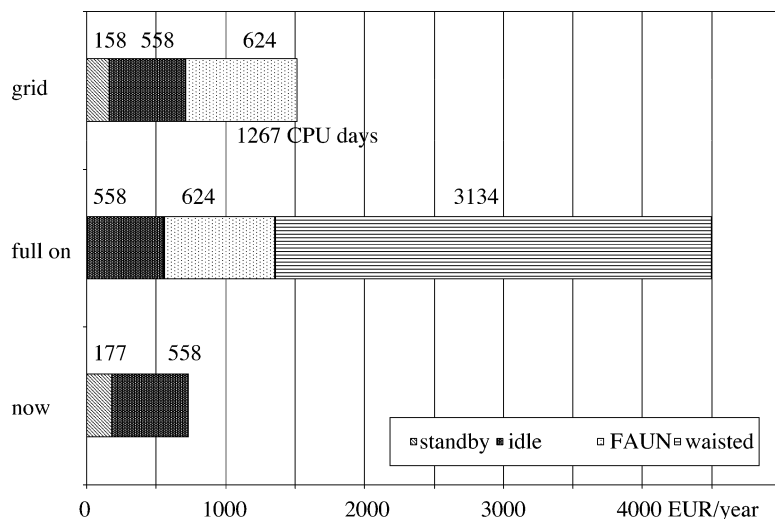


Figure 3: Power consumption (equivalent value in Euro) of the cluster with grid functionality and in full on status. *Standby* describes the power consumed by the computers simply being plugged. *Idle* means that the computers are on but do nothing. *FAUN* is the power used by FAUN computations. *Wasted* finally stands for the value of the current that would be drained if the computers were on all the year.

putting programs is to unify the computers to work together. See [2] for advanced links. Yet, a client providing grid computing functionalities as middleware is also suitable for distributed computing. The combination of distributed and grid computing used with parallel programming techniques is the target of the client. Distributed computing offers the opportunity to use computers all over the world with an easy to install client. Grid computing leads to error tolerance and an easy configurable system with advanced features. Up until now, usually only automatic wakeup and shutdown is concretely implemented.

For simple parallelization purposes specialized programming packages can be used. For general information, [22] and [10] are a good start. The most common packages are the Message Passing Interface (MPI) and the Parallel Virtual Machine (PVM); see [12, 16] for information. A first parallel version of the neurosimulator FAUN indeed uses PVM; see [9] and [8] for examples. But these packages are not suitable for programming a modular grid computing client.

The testbed for the grid computing client is shown in Fig. 4. It is decided that an individual protocol is developed for the grid computing client, which will connect to a central computing server. Known criteria for quality software are taken into account.

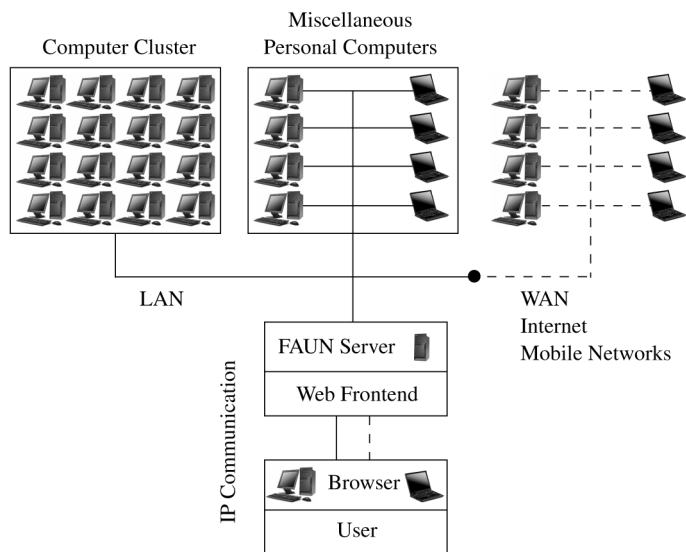


Figure 4: FAUN grid computing configuration.

A detailed description of the neurosimulator FAUN can be found in [6]; see also [5] and [25] for real life applications and [31, 32] for the last development steps. For information on differences between a coarse-grained and fine-grained parallelization see [8]. General introductions and more details are offered by [11, 16, 17, 19–21, 36, 43, 45]. The present parallelization of FAUN uses the master slave principle. The master program `faun_m` starts several instances of the slave program `faun_s`.

To get a good quality neural network it is necessary to compute a certain amount of neural networks. This amount varies with the difficulty of the problem. The quality of the neural network is primarily measured by the difference between the output as it is given by the neural network formula and the real output given by the training and validation data.

The specific amount of neural networks that will be calculated has to be determined heuristically. This amount also depends on the “difficulty” of the problem. After the computation the best network is chosen. This is the network that approximates the data better than all others. The neurosimulator uses a global optimization procedure with SQP optimization. Yet, it is not uncommon that difficult problems need long calculation times: One minute or more for every successfully trained neural network on a single personal computer, Pentium IV at 3.4 GHz.

The calculation time increases linearly with the amount of training and validation data. This is the initial reason why a parallel and now a grid computing version of

FAUN is developed. Computing times of one day or more are simply not acceptable especially for real time applications.

The parallelized version of FAUN based on the grid computing client runs without preliminary installation and offers additional features in comparison to the first version using PVM. The advantage of coarse-grained parallelization is common to all versions: a parallelization degree between 0.9 and 0.97. The meaning of this number is that if 100 computers work together, the computation will not be exactly 100 times as fast as with a single computer. But it will still be 90 – 97 times as fast as with a single computer. The loss of computing power can be explained with the additional administration and communication overhead¹.

A design goal coming together with the development of the grid computing client is to interface the parallel HPC version of FAUN to the already existing web interface for single processor computation developed by S. König. Now the web interface allows to manage the additional grid computing features, i.e., wake up and shutdown of the clients. It is then possible to decide which computers will be used for the computation. This offers the possibility that several users can share the resources simultaneously.

Another specialty has to be taken into account. The cluster of the School of Economics which is mainly used as testbed is protected with a firewall. That means that no access is possible from *outside* although the clients can connect from *inside* to arbitrary destinations. This is a problem when “sleeping” computers have to be woken up, because no inner access is available. A solution consists in leaving an intermediary computer with low power consumption on in the cluster. It will receive the wake up requests and send a magic packet to the desired computer.

4 Stackelberg Game and Neural Network Solution

In order to be able to compute a Stackelberg game, a sensible discretization has to be chosen. For the three control variables u_P , u_E , and η_E an appropriate set of controls has to be found. If $|U_P|$, $|U_E|$, and $|H_E|$ denote the number of elements in the respective set and s denotes the number of steps in the discretization, we have:

$$p = (|U_P| \cdot |U_E| \cdot |H_E|)^s \quad (14)$$

different strategies to choose from. As these strategies have to be computed for many different points to get good training data, it is wise to take small sets. There-

¹Imagine a bunch of potatoes that have to be peeled (famous “potato peeling problem”). A single person will work for a long time. If a second person helps the peeling will probably be twice as fast. But if more and more persons join in the peeling time will be spent on handing the potatoes from one to another and the entire process will be more and more inefficient. Finally, the largest potato determines the minimum time achievable.

fore, the smallest sets are chosen, which gives:

$$U_P = \{-1, 1\}, \quad (15)$$

$$U_E = \{-1, 1\}, \quad (16)$$

$$H_E = \{-1, 0.48\}. \quad (17)$$

Rightly, the reader may argue that this is too small a set for getting exact results. But two facts, besides the required computation time, have to be considered. First, sensible strategies seem to indicate that only extreme left/right turns or full deceleration/acceleration give good results. Second, the question has to be answered, which *exact* numbers should then be chosen to complete the sets. As the focus is on showing the opportunities of neural networks related to dynamic game, it seems acceptable to go with this simple approach. The choice of s is also dictated by the need for keeping the computation time small. Even with parallelization readily available an exponential increase can't be ignored. A viable approach is given with

$$s = 10, \quad (18)$$

although it is clear, that a higher value would have yielded a finer discretization. Stackelberg strategies are computed for 69800 points in the six-dimensional state space. For the computation parallelization is used, too. The strategies lead to 690000 points for which strategies exist because of (18). Every point in the state space belonging to a strategy can in fact be considered as being the initial set of another strategy.

Additionally, the strategies can be "mirrored" for P at the middle of the road, when the corresponding state variables, especially the angles, are also mirrored. This doubles the number of points so that approximately 1.4 million exist. The resulting maneuvers for five chosen start points can be seen in the upper row of Fig. 5. Even without a detailed analysis, it can be seen that the reactions are sensible and that the street boundary conditions are met. As positions are calculated by the display program from the center of the rectangle representing the car, it is acceptable for E to be off the road with *two* wheels. But the reader will notice that "half" of E is always on the road and that means that the street boundaries are honored.

It is now necessary to cut out training and validation data for the neurosimulator FAUN. As 1.4 million points would be too much, 50,000 points are taken. For this, the state space is equally divided in 100 hypercubes. These hypercubes are filled with 500 randomly chosen points of the appropriate region. The purpose of this procedure is to prevent the neurosimulator from overtraining one region and neglecting another. Finally, three hyperspheres are used as validation data. The ratio of training and validation data is 4 : 1 giving approximately 42,000 training and 8,000 validation inputs.

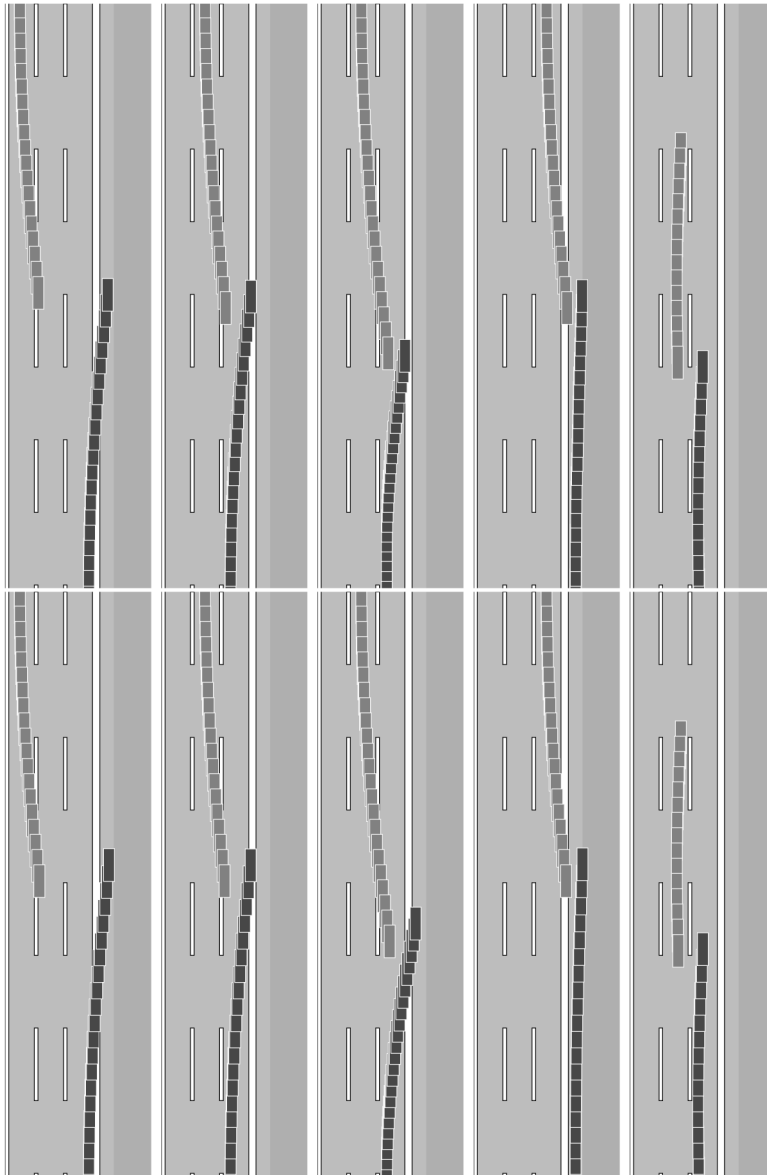


Figure 5: Trajectories for different start configurations of x_P and x_E . The first row shows the result from the Stackelberg simulation; in the second row neural network guidance is used. The results hardly differ. Pursuer coming from above (grey), evader coming from below (black).

In the case of neural networks parallelization can achieve a high speed up and shortens the computation time to *only* several hours in contrast to several days. For different parameter settings of the neurosimulator 1,000 neural networks are computed for each control variable and the best is taken. For instance, the resulting neural network for control variable u_E is given by the following lengthy result. It has to be kept in mind, though, that this formula can be computed in *real time* compared to the Stackelberg solution:

$$\begin{aligned}
u_E = & ((-1) + (2 * ((\tanh(((\tanh((3.36253335 \\
& + ((2 * ((x_P - 0.01061)/14.97877)) - 1) * 0.78373677) \\
& + (((2 * ((x_E - 0.01296)/14.97407)) - 1) * (-3.2545598)) \\
& + (((2 * ((y - 0.76436)/65.01923)) - 1) * (-0.2829929)) \\
& + (((2 * ((v_E - 14.04071)/40.37917)) - 1) * 0.02903405) \\
& + (((2 * ((\phi_P - 2.92346)/0.43626)) - 1) * (-0.14377207)) \\
& + (((2 * ((\phi_E - (-0.27143))/0.60569)) - 1) * (-2.4273138)))) \\
& * 7.3417816) + (\tanh((4.32403995 \\
& + (((2 * ((x_P - 0.01061)/14.97877)) - 1) * (-0.89812738)) \\
& + (((2 * ((x_E - 0.01296)/14.97407)) - 1) * 3.813688) \\
& + (((2 * ((y - 0.76436)/65.01923)) - 1) * (-0.3494907)) \\
& + (((2 * ((v_E - 14.04071)/40.37917)) - 1) * 0.065534994) \\
& + (((2 * ((\phi_P - 2.92346)/0.43626)) - 1) * 0.21560794) \\
& + (((2 * ((\phi_E - (-0.27143))/0.60569)) - 1) * 2.8371935))) \\
& * (-6.4195971) + (-0.42285788) \\
& + (((2 * ((x_P - 0.01061)/14.97877)) - 1) * (-5.0039272)) \\
& + (((2 * ((x_E - 0.01296)/14.97407)) - 1) * 5.531147) \\
& + (((2 * ((y - 0.76436)/65.01923)) - 1) * (-0.024304409)) \\
& + (((2 * ((v_E - 14.04071)/40.37917)) - 1) * 0.028052) \\
& + (((2 * ((\phi_P - 2.92346)/0.43626)) - 1) * 1.5422002) \\
& + (((2 * ((\phi_E - (-0.27143))/0.60569)) - 1) * 4.8255601))) \\
& + 0.95/1.9))). \tag{19}
\end{aligned}$$

In the context of this paper we don't want to explain the above formula in detail as an in-depth analysis of neural network formulas is furnished in [6]. For the reader it is more important to grasp its general significance. The formula is *simple* in the sense that besides the tanh function only elementary operators are used on the state variables: The first factor of the outer product on each line is only responsible for appropriate scaling. The second factor is the corresponding weight. The formula is thus *fast* to compute.

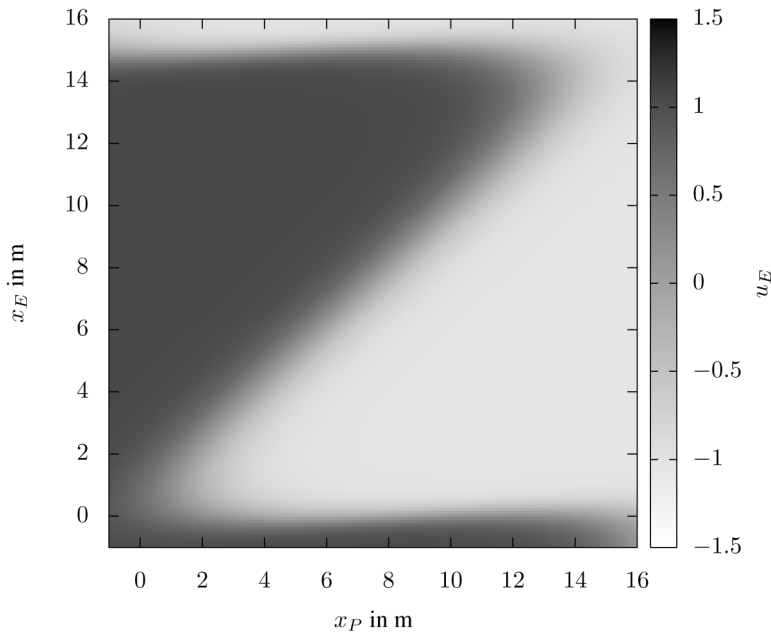


Figure 6: Resulting guidance u_E for different positions of x_P and x_E using neural networks ($y = 42\text{m}$, $v_E = 42\frac{\text{m}}{\text{s}}$, $\phi_P = \pi$, $\phi_E = 0$). The Z-shape marks the dispersal surfaces of this game.

As can be seen on Fig. 5 which compares Stackelberg game and neural network guidance, the results hardly differ “optically”. And, indeed, a more detailed analysis at some exemplary points yields good results. Figure 6 shows the result of neural network guidance for the control variable u_E . It is clear that E has to steer left when he is left of P and right when he is right of P. Therefore, a dispersal surface at the diagonal for $x_P = x_E$ can be expected and is indeed reproduced by the neural network. Continuous colors are added for better legibility of the plot. Normally the output of the neural network would be mapped to ± 1 as this is, what is trained with the Stackelberg game. Figure 6 also shows that the boundary constraints are observed, leading to the typical Z-shape of the figure. Indeed, for $x_E \approx 0\text{m}$ and $x_E \approx 15\text{m}$ E has to steer in the opposite direction to stay on the freeway. Of course, this brings him nearer to P and thus to capture.

The occurring error is analyzed in more detail in Fig. 7. A point is set if the sign of the neural network output is opposite to the Stackelberg game. Errors only happen at the dispersal surfaces and don’t reach far. This means, that a little away from the dispersal surface the sign is correct. A small error on the diagonal doesn’t

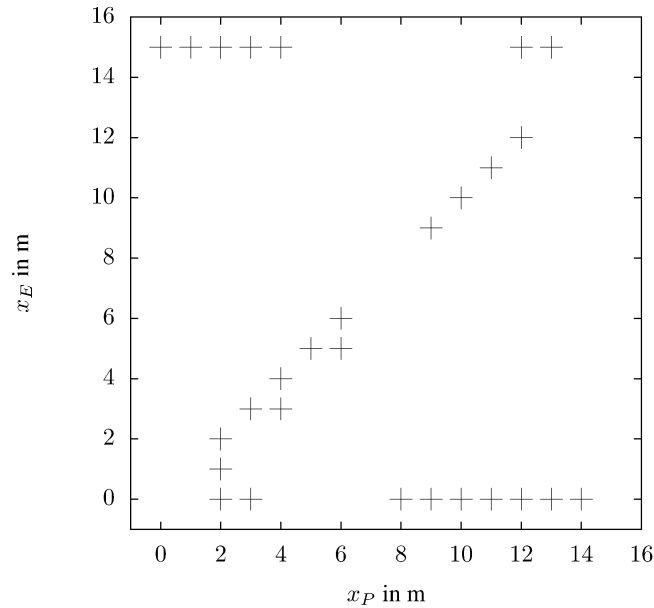


Figure 7: Occurring errors for u_E . A point is set, if the neural network would steer in the opposite direction of the Stackelberg solution. This happens only on the diagonals, where it doesn't matter much and on the limits of the road for x_E where the boundary conditions have to be met ($y = 42\text{m}$, $v_E = 42\frac{\text{m}}{\text{s}}$, $\phi_P = \pi$, $\phi_E = 0$). Figure 10 complements this qualitative error plot with absolute values.

harm much as this constellation is very bad for E anyway. Other small errors can be observed at the street boundaries, where the neural network guidance sometimes reacts too early and sometimes too late as will be explained in more detail on Fig. 8. Note, that in fact errors at the street boundaries are of marginal importance as this could also be controlled by other means. Indeed, today's trucks are already often equipped with special devices that warn the driver, if the truck leaves its lane.

Figure 9 shows the value of the game. In this case, the time until capture is not taken, but merely the minimal distance between P and E, because it is the payoff which divides between accident (capture from P) and unharmed drive (evasion of E). The actual minimum distance for evasion can be set adequately but $d = 1.5\text{m}$ seems a good approximation for a normal car. Figure 9 has to be read as follows: black areas signalize capture in other cases E can evade. The upper plot makes the whole game look satisfactorily for E as capture only occurs when P and E are almost face to face on the freeway. But this doesn't have to always be like that. The good results for E can only be achieved because of the relatively small $y = 42\text{m}$ at the beginning of the game in the upper plot.

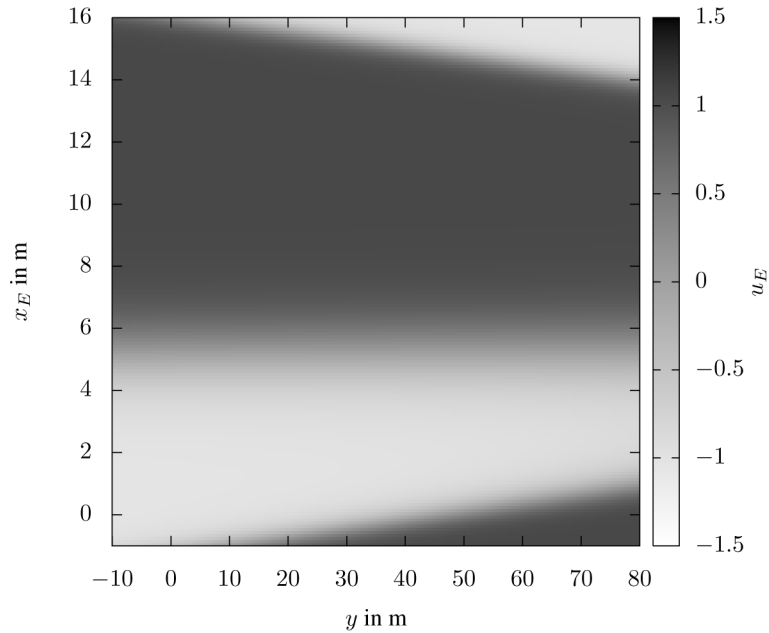


Figure 8: Neural network guidance for different values of y and x_E , while $x_P = 5\text{m}$ fixed ($v_E = 42\frac{\text{m}}{\text{s}}$, $\phi_P = \pi$, $\phi_E = 0$). The guidance is (almost) error free on the road. The (hard) boundary conditions are not always honored for small distances of y , but this doesn't change the outcome of the game much.

If this distance is significantly greater P can achieve capture. We have to consider that at least E is bounded by the width of the road. The Game of Two Cars therefore offers disadvantages for E compared to a game where, for instance, the whole xy -plane can be played on. Small distances are taken, because on a normal freeway the distance drivers and possibly collision avoidance gear can overview are not too long. Only for small distances are collision avoidance devices of interest, because they close the gap which results from the human capacity of reaction. A gloomier outlook for E can be seen in the lower part of Fig. 9. The initial distance is approximately twice as large as in the upper plot and the speed at the begin of the game is also reduced. When both actors start in the middle of the road nothing changes much. But when P and E are both near the border of the same side of the freeway the situation worsens for E as P has more time (and more space for maneuvers) to close in on E . If the initial distance between P and E is larger the situation degrades even more. A similar worsening can also be noticed for smaller starting velocities v_E of E , although a slight compensation happens due to the increased possibility for maneuvers. Indeed, the angular velocity w_E depends

on E as shown in (12). In certain situations a speed reduction can therefore be interesting, see below.

Figure 10 analyzes the case in more detail. The difference between the Stackelberg and the neural network result is shown. As can already be expected from the qualitative Fig. 7, the biggest errors happen on the dispersal surfaces. On the other hand, it has to be noticed that the errors are small in comparison to the width of the street. For the biggest error the inequality:

$$\Delta_{\max} < \frac{0.7\text{m}}{15\text{m}} \approx 0.047$$

holds. The average error is significantly smaller:

$$\Delta_{\text{avg}} < \frac{0.2\text{m}}{15\text{m}} \approx 0.013.$$

Most parts of the diagonal are within Δ_{avg} and therefore the outcome for E is not much worse using neural network guidance compared to the Stackelberg game.

Finally, the resulting dependencies of u_E using neural network guidance for various distances y is plotted in Fig. 8. Deliberately the initial position of P is set to $x_P = 5\text{m}$. As might be expected, a dispersal surface is clearly seen at $x_E = 5\text{m}$. However, the behavior at the boundaries of the freeway is surprising. The “rough direction” is right, but the neural network seems to react too early for large distances y and too late for small distances. This is a disadvantage but can be explained by the distribution of the training data. Although the distribution is smoothed by cutting out hypercubes it is nevertheless the case, that much more points have been calculated for small distances y as these are the most interesting. For larger distances ($y \approx 80\text{m}$) fewer points have been calculated and very few are on the border. The frequency of boundary points is low for large distances y and high for small distances. As neural networks are continuous functions the results are exaggerated for small and large y . A better fine tuning of the training parameters for the neurosimulator might improve the behavior and will be explored.

Up to the emphasis of the discussion has been on the control variables u_E and u_P — the steering directions of E and P. However, the game comprises a third control η_E being the velocity change rate. Figure 11 shows that acceleration or deceleration is sensible for different initial ratios of $v_E(t_0) : v_P$. First, a dispersal surface can be found at $v_E(t_0) = v_P (= \text{const.})$. In the neighborhood of this boundary E will accelerate if $v_E(t_0) > v_P$ and decelerate if $v_E(t_0) < v_P$, the regions marked *B* and *C*, respectively. (The singular case $v_E(t_0) = v_P$ is not analyzed here.) This can be explained by the fact that in both cases an advantage can be gained. Accelerating reduces the time until $y = 0$ and thus the time P has for maneuvering. In contrast, by decelerating E increases his maneuverability but also the total time of the game. If, e.g., E accelerates with $v_E(t_0) < v_P$ the disadvantage of the reduced maneuverability doesn't outweigh the advantage of the total game time reduction and vice versa.

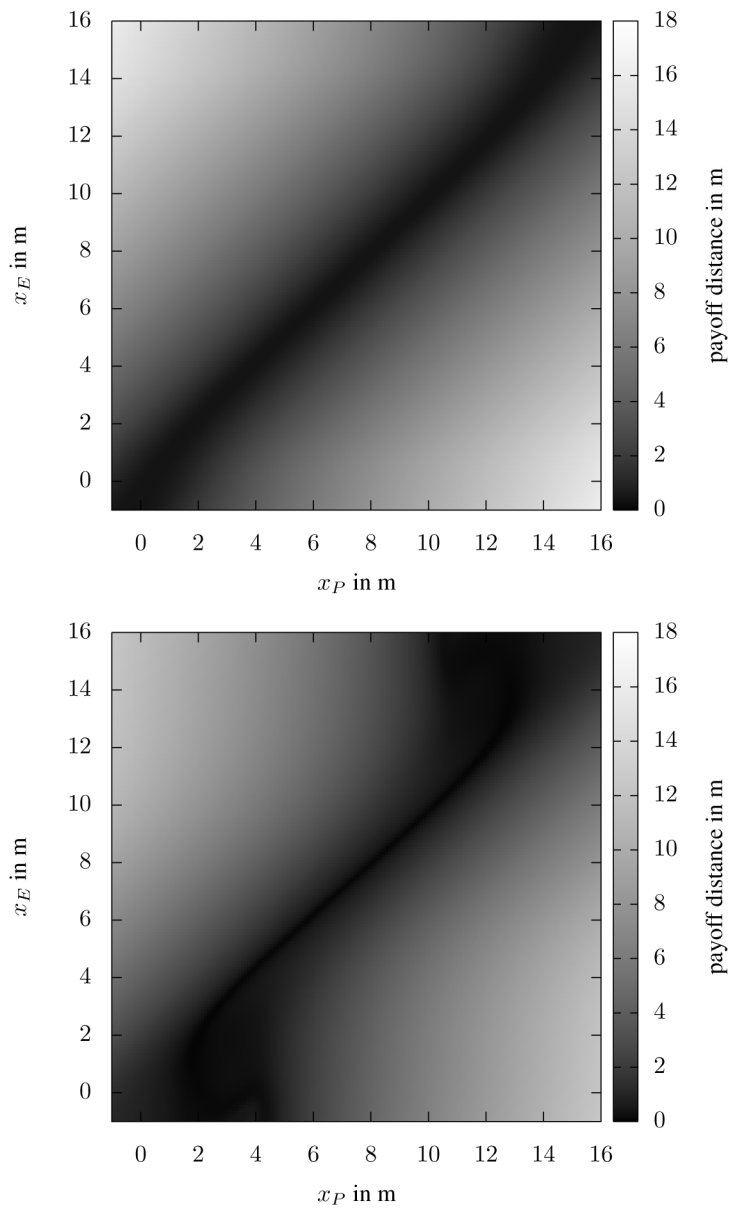


Figure 9: Payoff distance. Upper plot: $y = 42\text{m}$, $v_E = 42\frac{\text{m}}{\text{s}}$, $\phi_P = \pi$, $\phi_E = 0$. Lower plot: $y = 80\text{m}$, $v_E = 27\frac{\text{m}}{\text{s}}$, $\phi_P = \pi$, $\phi_E = 0$. Black color signals that a collision cannot be avoided.

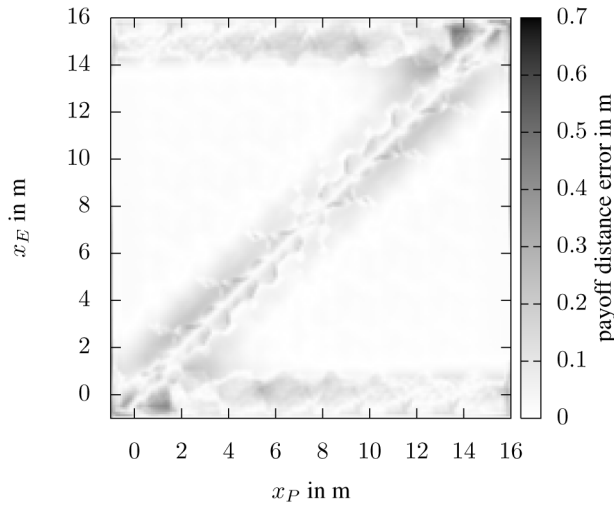


Figure 10: Payoff distance error: $y = 42\text{m}$, $v_E = 42\frac{\text{m}}{\text{s}}$, $\phi_P = \pi$, $\phi_E = 0$. The typical Z-shape of the dispersal surface appears. The error stays mostly below 0.2m . The pattern which can be seen in the low error regions is due to the interpolation through neural networks, which will deviate from the Stackelberg solution in the areas where no training data are available.

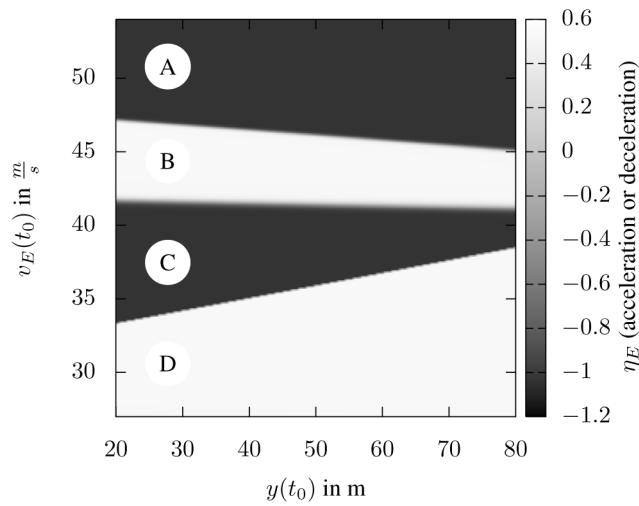


Figure 11: η_E for fixed $x_E(t_0) = 11\text{m}$, $x_P(t_0) = 4\text{m}$. Negative values signify that E is decelerating while positive values show an acceleration. The dispersal surface at $v_E(t_0) = v_P$ is noticeable.

There is, however, a sharply delimited dispersal surface where it again makes sense for E to decelerate (even if $v_E(t_0) > v_P$) or accelerate (even if $v_E(t_0) < v_P$), regions *A* and *D*. The greater the initial distance $y(t_0)$, the sooner this inversion happens. Qualitatively, both phenomena are understandable. On the one hand, the disadvantages of reduced maneuverability resp. velocity cannot be negated. On the other hand, for short distances and hence short game duration, the choice of η_E is more persistent and an inverse strategy only occurs for very high or very low velocities. It is clear that this qualitative analysis is unsatisfying and that further research will be dedicated to a solid explanation of Fig. 11.

The question remains whether the resulting neural network can be implemented as onboard system. The problems of positioning in the state space apart: Eq. (19) consists of 3 hyperbolic tangents, 39 multiplications, 19 divisions, 22 additions, and 50 subtractions. The hyperbolic tangents are expensive in terms of floating point operations and count for 10. This gives 160 floating point operations for one control. As E has two controls a total of 320 operations follows. Even if 1000 computations occur per second, i.e., $320 \cdot 10^3$ operations, modern processors easily manage $3 \cdot 10^9$ operations per second.

5 Conclusion and Outlook

Artificial neural networks are successfully employed to synthesize optimal strategies for the enhanced game of two cars. This game is much more realistic than the original game of two cars because constraints keep the correct driver on the freeway lanes always.

The usage of grid/distributed computing offers high-performance, low-budget computing (often at no costs for idle computers). Idle PCs are typical for public or company PC/computer clusters, e.g., at the Hanover School of Economics 38 modern PCs are idle during every night and on weekends. Software quality requirements increase: Issues like security and integrity become more important and moreover user friendliness, maintainability, and secrecy (user shouldn't notice computations on their office PCs). Grid/distributed computing is of particular relevance for all tasks of coarse-grained parallelization where interprocessor communication speed is of minor importance for the overall performance.

The neurosimulator FAUN repeats analogous tasks many times in order to train many artificial neural networks. Thus FAUN is well suited for distributed computing and a coarse-grained parallelization cuts down computing times significantly. Problems like the enhanced game of two cars which need computing days or even weeks on a standard PC can be solved in few hours. FAUN can also be used, e.g., for the solution of other (pursuit-evasion) dynamic games, aerospace guidance problems or forecasting problems of exchange or interest rates.

Nevertheless the usage of artificial neural networks is no easy to use black box method as often assumed. Problem analysis and formulation and also training necessitate deep knowledge. Usually knowledge about the special application is

important, too. With the distributed computing version of FAUN the user friendliness increases as usage is possible via a graphical web frontend. No manual editing of configuration files is necessary.

The developed grid computing client is not limited to the neurosimulator FAUN. Modularity enables the usage with other programs which can therefore benefit from automatic client updates and remote power management. Future development aims at improved security, reduction of bandwidth requirements, and better automation of installation and management.

The application of artificial neural networks to general dynamic games is very promising. It cannot replace a thorough mathematical analysis (and a solution where possible) but for many problems not (fully) solvable a fast and reliable numerical approximation of the solution often becomes possible. Six state and three control variables are involved here but even higher-dimensional dynamic games can be solved numerically using artificial neural networks and a high-end neurosimulator.

References

- [1] A. Abbas. *Grid Computing: A Practical Guide to Technology and Applications*. Charles River Media, Boston, 2004.
- [2] H. Attiya and J. L. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley-Interscience, New York, 2004.
- [3] T. Başar and G. J. Olsder. *Dynamic Noncooperative Game Theory*. Academic Press, London, 1995.
- [4] P. Borowko and W. Rzymowski. On the game of two cars. *Journal of Optimization Theory and Applications*, 44(3):381–396, November 1984.
- [5] M. H. Breitner. Robust optimal onboard reentry guidance of a space shuttle: Dynamic game approach and guidance synthesis via neural networks. *Journal of Optimization Theory and Applications*, 107:484–505, 2000.
- [6] M. H. Breitner. *Nichtlineare, multivariate Approximation mit Perzeptrons und anderen Funktionen auf verschiedenen Hochleistungsrechnern (in German)*. Akademische Verlagsgesellschaft, Berlin, 2003.
- [7] M. H. Breitner. Usage of artificial neural networks for the numerical solution of dynamic games. In T. L. Vincent, editor, *Proceedings of the Eleventh International Symposium on Dynamic Games and Applications, Tucson, Arizona*, volume 1, pages 62–79. University of Arizona Press, 2004.
- [8] M. H. Breitner, P. Mehmert, and S. Schnitter. Coarse- and fine-grained parallel computation of optimal strategies and feedback controls with multilayered feedforward neural networks. In A. Nowak, editor, *Proceedings of the Ninth*

International Symposium on Dynamic Games and Applications, Adelaide, Australia, 2000.

- [9] M. H. Breitner and H.-J. v. Mettenheim. Coarse-grained parallelization of the advanced neurosimulator FAUN 1.0 with PVM and the enhanced cornered rat game revisited. *International Game Theory Review*, 7:347–365, 2005.
- [10] J. Dongarra, I. Foster, G. C. Fox, W. Gropp, K. Kennedy, L. Tonczon, and A. Whithe. *Sourcebook of Parallel Computing*. Morgan Kaufmann, Amsterdam, 2003.
- [11] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, New York, 2000.
- [12] G. A. Geist, A. L. Beguelin, J. Dongarra, W. C. Jiang, R. J. Manchek, and V. S. Sunderam. *PVM: Parallel Virtual Machine – A Users’ Guide and Tutorial for Networked Parallel Computing*. MIT-Press, Cambridge, 1994.
- [13] W. M. Getz. Capturability in a two target ‘game of two cars’. *Journal of Guidance, Control and Dynamics*, 4(1):15–21, 1981.
- [14] W. M. Getz and M. Pachter. Two-target pursuit-evasion differential games in the plane. *Journal of Optimization Theory and Applications*, 34(3):383–403, July 1981.
- [15] I. Greenfeld. A differential game of surveillance evasion of two identical cars. *Journal of Optimization Theory and Applications*, 52(1):53–79, January 1987.
- [16] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI*. MIT-Press, Cambridge, 1999.
- [17] X. Gui, Q. Wang, and D. Quian. A grid middleware for aggregative scientific computing libraries and parallel programming environments. In *Proceedings of the 6th Asia-Pacific Web Conference, Hangzhou*, pp. 671–676. Springer, Heidelberg, 2004.
- [18] A. J. G. Hey, G. Fox, and F. Berman. *Grid Computing*. John Wiley & Sons, Chichester, 2003.
- [19] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257, 1991.
- [20] K. Hornik, M. Stinchcombe, and H. Whithe. Multi-layer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.
- [21] K. Hornik, M. Stinchcombe, and H. Whithe. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3:551–560, 1990.

- [22] C. Hughes and T. Hughes. *Parallel and Distributed Programming Using C++*. Addison-Wesley, Boston, 2003.
- [23] R. Isaacs. *Differential Games – A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. John Wiley & Sons, New York, 1965 – reprint.
- [24] J. Joshy and C. Fellenstein. *Grid Computing*. Prentice Hall PTR, London, 2003.
- [25] F. Köller and M. H. Breitner. Efficient synthesis of strategies with the advanced neurosimulator FAUN 1.0. In M. H. Breitner, editor, *Proceedings of the Fourth International ISDG Workshop, Goslar, 2003*.
- [26] R. Lachner. *Echtzeitsynthese optimaler Strategien für Differentialspiele schneller Dynamik mit Anwendungen bei der Kollisionsvermeidung (in German)*. PhD thesis, Technische Universität Clausthal, Clausthal-Zellerfeld, 1997.
- [27] R. Lachner, M. H. Breitner, and H. J. Pesch. Real-time collision avoidance: Differential game, numerical solution, and synthesis of strategies. In J. A. Filar, V. Gaitsgory, and K. Mizukami, editors, *Advances in Dynamic Games and Applications*. Birkhäuser, Boston, 2000.
- [28] J. P. Marec and N. V. Nhan. Two-dimensional pursuit-evasion game with penalty on turning rates. *Journal of Optimization Theory and Applications*, 23(2):305–345, October 1977.
- [29] A. W. Merz. *The Homicidal Chauffeur – A Differential Game*. PhD thesis, Department of Aeronautics and Astronautics, Stanford University, Stanford, 1971.
- [30] A. W. Merz. The game of two identical cars. *Journal of Optimization Theory and Applications*, 9(5):324–343, May 1972.
- [31] H.-J. v. Mettenheim and M. H. Breitner. Neural network forecasting with high performance computers. In E. P. Hofer and E. Reithmeier, editors, *Proceedings of the Thirteenth International Workshop on Dynamics and Control*, pp. 33–40. Shaker, Aachen, 2005.
- [32] H.-J. v. Mettenheim and M. H. Breitner. Distributed neurosimulation. In H.-D. Haasis, H. Kopfer, and J. Schönlberger, editors, *Operations Research Proceedings*. Springer, Heidelberg, 2006.
- [33] A. Meyer, M. H. Breitner, and M. Kriesell. A pictured memorandum on synthesis phenomena occurring in the homicidal chauffeur game. In G. Martin-Herran and G. Zaccour, editors, *Proceedings of the Fifth International ISDG Workshop, Segovia*, pp. 17–32, 2005.

- [34] T. Miloh. The game of two elliptical ships. *Optimal Control Applications & Methods*, 4, 1983.
- [35] S. Mukherjee, J. Mustafi, and A. Chaudhuri. Grid computing: The future of distributed computing for high performance scientific and business applications. In *Distributed Computing. Mobile and Wireless Computing. 4th International Workshop, IWDC*. Springer, Heidelberg, 2002.
- [36] B. Müller, J. Reinhardt, and M. T. Strickland. *Neural Networks*. Springer, Berlin, 1995.
- [37] M. Pachter and W. M. Getz. The geometry of the barrier in the game of two cars. *Optimal Control Applications and Methods*, 1:103–118, 1980.
- [38] J. Shinar and A. Davidovitz. New results in the game of two identical cars. In *Analysis and Optimization of Systems*, volume Volume 83/1986, pp. 759–780, 1986.
- [39] J. Shinar and R. Tabak. New results in optimal missile avoidance analysis. *Journal of Guidance, Control & Dynamics*, 17, 1994.
- [40] W. R. Stevens, B. Fenner, and A. M. Rudoff. *UNIX Network Programming - The Sockets Networking API*, volume 1. Addison-Wesley, Boston, 2004.
- [41] W. R. Stevens and S. A. Rago. *Advanced Programming in the UNIX Environment*. Addison-Wesley, Upper Saddle River, NJ, 2005.
- [42] A. E. Utemov. Numerical control optimization methods in one pursuit-evasion problem. *Journal of Computer and Systems Sciences International*, 45(3):395–412, May 2006.
- [43] V. R. Vemuri. *Artificial Neural Networks*. IEEE Computer Society Press, Los Angeles, 1994.
- [44] T. L. Vincent, E. M. Cliff, W. J. Grantham, and W. Y. Peng. Some aspects of collision avoidance. *AIAA Journal*, 12, 1974.
- [45] H. Whitte, A. R. Gallant, K. Hornik, M. Stinchcombe, and J. Wooldridge. *Artificial Neural Networks – Approximation and Learning Theory*. Blackwell, Oxford, 1992.
- [46] Y. Yavin and R. de Villiers. Game of two cars: Case of variable speed. *Journal of Optimization Theory and Applications*, 60(2):327–339, February 1989.
- [47] Y. Yavin and R. de Villiers. Proportional navigation and the game of two cars. *Journal of Optimization Theory and Applications*, 62(3):351–369, September 1989.

Annals of the International Society of Dynamic Games

Advances in Dynamic Games and Their Applications Analytical and Numerical Developments

Pierre Bernhard, Vladimir Gaitsgory, and Odile Pourtallier
Editors

This book—an outgrowth of the 12th International Symposium on Dynamic Games—presents current advances in the theory of dynamic games and their applications in several disciplines. The selected contributions cover a variety of topics ranging from purely theoretical developments in game theory, to numerical analysis of various dynamic games, and then progressing to applications of dynamic games in economics, finance, and energy supply.

Thematically organized into eight parts, the book covers key topics in these main areas:

- theoretical developments in general dynamic and differential games
- pursuit-evasion games
- numerical approaches to dynamic and differential games
- applications of dynamic games in economics and option pricing
- search games
- evolutionary games
- stopping games
- stochastic games and “large neighborhood” games

A unified collection of state-of-the-art advances in theoretical and numerical analysis of dynamic games and their applications, the work is suitable for researchers, practitioners, and graduate students in applied mathematics, engineering, economics, as well as environmental and management sciences.

Birkhäuser
www.birkhauser.com

