International Journal of Innovative
Computing, Information and Control
Volume **15**, Number **5**, October **2019**

 ISSN 1349-4198
pp. 1747–1761

CAPTURING THE BRACHISTOCHRONE: NEURAL NETWORK SUPERVISED AND REINFORCEMENT APPROACHES

Raed Abu Zitar

Department of Computer Engineering
Ajman University
University Street, Al Jerf-1, P.O.Box: 346, Ajman, United Arab Emirates
r.abuzitar@ajman.ac.ae

Received February 2019; revised June 2019

Abstract. *This paper attempts to answer the question regarding the capability of neural networks (NN) to capture an optimum control problem for finding a minimum time path constrained problem. It is called the Brachistochrone. This problem is a famous calculus of variations problem seeking the optimization of an integral that has a solution in the form of a curve or a two-dim function. The proposed neural network (NN) model did not only imitate the learning samples but could capture and understand the law of control (surface) for this benchmark problem as we will see in the paper. The genetic algorithm (GA) is used in extracting many solutions for the problem at different initial conditions. Samples of solution extracted by the genetic algorithm (GA) in addition to some solutions found by a classical numerical optimization algorithm (Nelder-Mead) are used in training a neural network in a supervised model. Another neural network with a different architecture, the grid architecture, and a new reinforcement-learning algorithm is used to train the neural network to capture the Brachistochrone. This time no supervisor is used in training. Results of simulations are presented, comparisons are made and promising findings are reached that indicate the capabilities of different neural networks paradigms to learn and capture optimum control strategies.*

Keywords: Brachistochrone, Calculus of variations, Genetic algorithm, Neural networks, Reinforcement learning

1. **Introduction.** The Brachistochrone is an old calculus of variations problem, please see Figure 1. It is the Johann Bernoulli's famous 1696 problem [1]. Recently, we are in 2019 and we are still interested in this genuine problem that attracted the attention of many researchers over the years. It simply can resemble a calculus of variations problem, differential geometrics problem, and/or optimal control problem. The challenge in this problem is to find the shortest path in time for an object sliding between two points under the effect (constraint) of gravity. The answer to this challenge is "No, the shortest path between two points is not the straight line". It may be looked at as an optimum control problem where the minimum time path between two points under the constraint of gravity is sought. The initial velocity could be variable; however, the particle slides with no force affecting it except the force of gravity. In this work, we are not only interested in finding the solution of this problem in different ways, but also in exploiting the capabilities of an artificial intelligence paradigm such as neural networks in "understanding" and capturing the control strategy of the Brachistochrone with learning examples and even without learning examples. The solution has been found analytically using the calculus of variations. Thorough and detailed analysis exists in [2,3]. Many authors considered it as an optimum control problem as in [4]. We call those classical analytical methods.

DOI: 10.24507/ijicic.15.05.1747

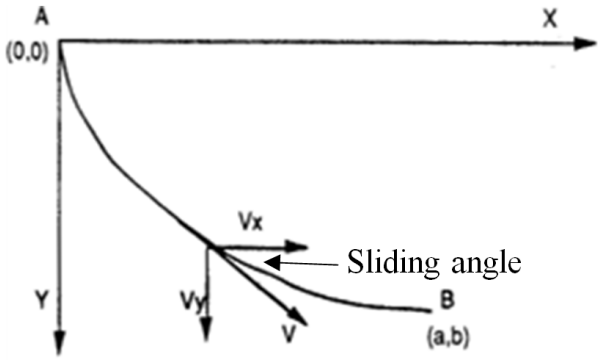

FIGURE 1. Path of the sliding particle

The problem was also solved with numerical methods and with evolutionary optimization methods [5-7]. All those techniques relied on discovering a set of support locations in the two dimensional space based on spanning the space of possible path solutions. What we are interested in here is "Can the Neural networks (NN) generalize the solution for this temporal space problem?" The cost function here is an integral and the solution is a path or a function of x and y. Can NNs capture this control surface for any given initial condition? We will investigate two cases. The first one is the ability of NN to learn this control surface by presenting samples of solution extracted by the genetic algorithm (GA), we call this algorithm "The Neural Network – GA Surface Approximation Algorithm". The second one investigates the ability of NNs to learn on its own strategy of optimum control with simple reinforcement rules and a parallel-distributed architecture (grid). The two techniques, although can be implemented by the NN, they are very different in approach. The first one is a supervised learning technique which interpolates and extrapolates samples of training solutions found with different initial conditions. The other technique is based on continuous reinforcement (reward/penalty) actions done locally on the nodes with global selection method that covers the whole search space. The cost in all cases is based on a function of the sequence of points that resembles the suggested path. Both techniques, by the way, can also be applied on different optimization problems that seek a pattern of values or sequence of temporal actions.

The genetic algorithms (GA) are class of optimization algorithms that mimic the mechanics of real life genetics [9,10]. Since the minute they were introduced, fourty years ago, they are being exploited in different optimization and machine learning problems. Genetic algorithms (GA) are general purpose optimization algorithms with guided probabilistic components [11-13]. The genetic algorithm was used in solving the Brachistochrone problem before [5,6]. It was used to search the two-dimensional horizontal and vertical space for a sequence of points that make up a possible path of minimum descending time. The solution for the Brachistochrone is an integral that uses functions of first order and second order derivatives of the $y(x)$ function, please see Figure 1. The problem may have many applications in transportation, flights cruise control, roads and bridges design, and even in military and defense.

Figure 1 presents the Brachistochrone which is the minimum time path between two fixed points under gravity. The first point is at 100 ft height and the other point is at 0.00 ft height. The horizontal distance between the two points is 100 ft. The figure shows the path of minimum time under the initial velocity (V_o) of 0.0 ft/sec.

To analyze and optimize the path followed by a particle under the influence of gravity, analytical methods can be used to solve this problem; however, they all require well defined continuous functions and derivatives of first and second order [2,3]. From the physics of

the problem, $y(x)$ must obey certain restrictions: not only must it be continuous but it must also be twice differentiable since the existence of (dy/dx) is directly related to the existence of the acceleration of the particle. The function $y(x)$ must also satisfy the boundary conditions:

$$y(0) = 0, \quad y(a) = b \tag{1}$$

Find the extremum of the functional T where:

$$T = \int F(y, y', x)dx \tag{2}$$

We must use Euler-Lagrange equations [3] and the solutions of these equations give stationary point for T, which may be global or local minimum (one of the classical method disadvantages). Since all partial derivatives of F are functions of y, y', and x, this DE involves no higher derivatives of y than the second derivative. The analysis of the Brachistochrone problem by the Euler-Lagrange equation after some manipulations yields the following DE:

$$2y'y + 1 + y'2 = 0 \tag{3}$$

The standard substitution and integration of Equation (3) yield a solution for y and x in parametric form as:

$$x = k/2(\emptyset - \sin \emptyset) \tag{4}$$

$$y = k/2(l - \cos \emptyset) \tag{5}$$

$\emptyset$ is a polar parameter.

These equations represent a cycloid which actually yields a minimum for T. The numerical solution to this problem is presented in the next section. Classical numerical methods were also used to solve this problem such as the Nelder-Mead algorithm [5,6]. However, those classical methods have limitations on the number of variables and the associated complexity. The design variables in this problem are the intermediate support heights, and the start and the end points are fixed.

In conclusion, the Brachstochrone problem was solved analytically (using calculus of variations and optimum control theory), numerically (Nelder-Mead algorithm), and using evolutionary based algorithms such as genetic algorithms (GA). Furthermore, neural networks (NN) were also used in solving this problem in previous studies [14]. They used NN to map the location of the particle with the angle of the sliding particle. The NN is used to interpolate a value for the angle based on samples of existing solutions for the support locations (x, y). The solution provided by this study is based on local observations for the location of the particle. It did not show that the NN have captured a general low of control, and it is more like a demonstration for the interpolation capabilities of NN. The study in [15] follows a different approach. It considers one support location x as input and the second support location y as output. A feedforward NN is used to generate the output based on a given input. A cost function is formulated for the trajectory time. The method relies on finding Jacobian matrix and assuming possible solutions for the path. Back propagation algorithm is used in the training of the NN. The method relies more on numerical techniques to optimize the cost function. The NN is used only to generate the path. Again the architecture and the learning capabilities are not tested here for capturing a general control law that provides solutions for the Brachistochrone problem.

The following points can summarize the road map for the rest of this paper.

1) In Section 2, the GA and Nelder-Mead are used in extracting solutions for the Brachstochrone under different parameters values. Comparisons and analysis are made.

2) In Section 3, the rules extracted by the GA are used in training a feedforward NN to build a compact controller that can solve this benchmark problem. It is a supervised learning method.

3) In Section 4, the grid based NN reinforcement algorithm is used to generate solutions for the same problem without any priori knowledge or supervised learning. It is unsupervised learning method.

4) In Section 5, discussions and analysis of simulation results are presented.

5) In Section 6, conclusions and future work are presented.

2. **The Genetic Algorithm (GA).** Optimization by GAs is different from other search methods encountered in engineering in the following ways.

i) GAs require parameter set of the optimization problem to be coded as a finite-length string over some finite alphabet.

ii) GAs search simultaneously from a population of strings, climbing many picks in parallel, therefore, reducing the problem of a false pick.

iii) GAs use probabilistic rules rather than deterministic rules to guide their search. The use of probabilistic rules does not suggest that the method is a random search method.

A simple genetic algorithm consists of three operators. 1) Reproduction – individuals contribute to the gene pool in proportion to their relative fitness (evaluation on the function being optimized). Thus, well performed individuals contribute multiple copies to gene pool. 2) Crossover – select two parent at random from the gene pool as well as a crossover point within the binary encoding. The parents exchange 'tails', that is, the portion of the string to the right of the crossover point to generate two offspring. 3) Mutation – alteration of a string position with a low probability. Mutation is used to preserve the population diversity. We applied the genetic algorithm to the Brachistochrone problem. The goal here is to connect the GA with the existing optimization and control method literature.

As stated earlier, one of the necessary conditions for using the GA is the ability to code the underlying parameter set as finite length string (i.e., individual). In this problem, trade-off has been made between the number of support locations and the string length to achieve desirable descent time and faster convergent rate. Thus, the full string is formed from concatenation of 4 ten-bit substrings where each substring is mapped to fixed point support location. For testing GA, it is assumed that the height is at 100 ft above the ground with resolution factor of 0.1 on each of the four support locations. Therefore, the space searched by the GA consists of $(1024) * 4$ alternative solutions. To initiate simulations, starting populations of 25, 200, and 400 strings are selected at random.

For each run of the GA trials, we run to generation 500. These represent a total of $25 * 500$, $200 * 500$, and $400 * 500$ function evaluations per independent trial. The results from these trials are shown in Figure 2. The minimum descent time is 3.49 sec, 3.44 sec, and 3.41 second for the population size of 25, 200 and 400, respectively. The maximum difference in minimum descent time does not exceed more than .08 seconds, this small difference is because once the fitness function reaches convergence after the maximum generations of 500, then the values of minimum time will be almost identical regardless of the population size which is critical only at the beginning.

Figure 3 depicts the solution for frictionless case extracted by Nelder-Mead algorithm, a classical optimization algorithm suitable for pattern search. The time taken by this is 3.244 sec similar to our analytical results. For additional details and examples with friction factors, the reader is referred to [5]. The biggest problem with this type is that it cannot handle more than 6 support locations in the path. We will be relaying on the genetic algorithm (GA) in extracting solutions for this work.

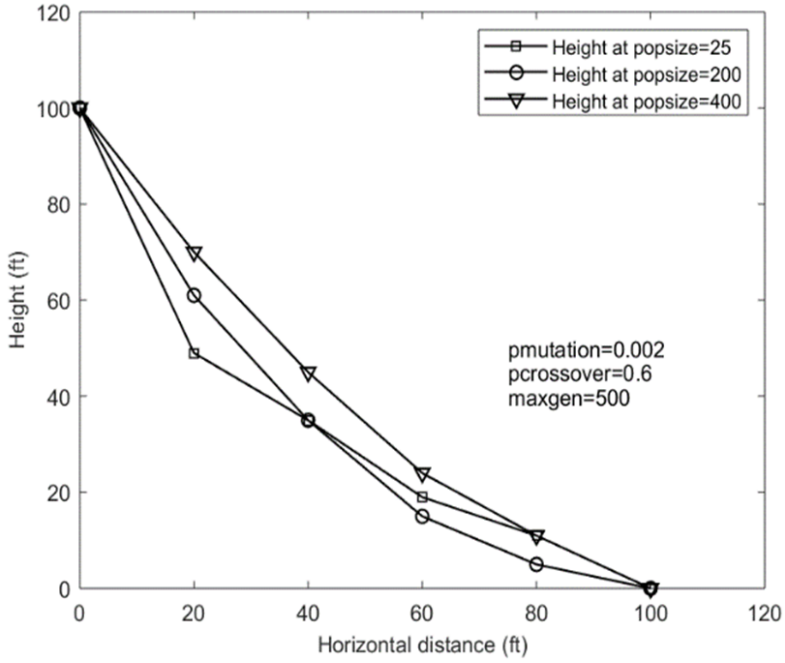

FIGURE 2. Optimal support locations found by GA with different population sizes

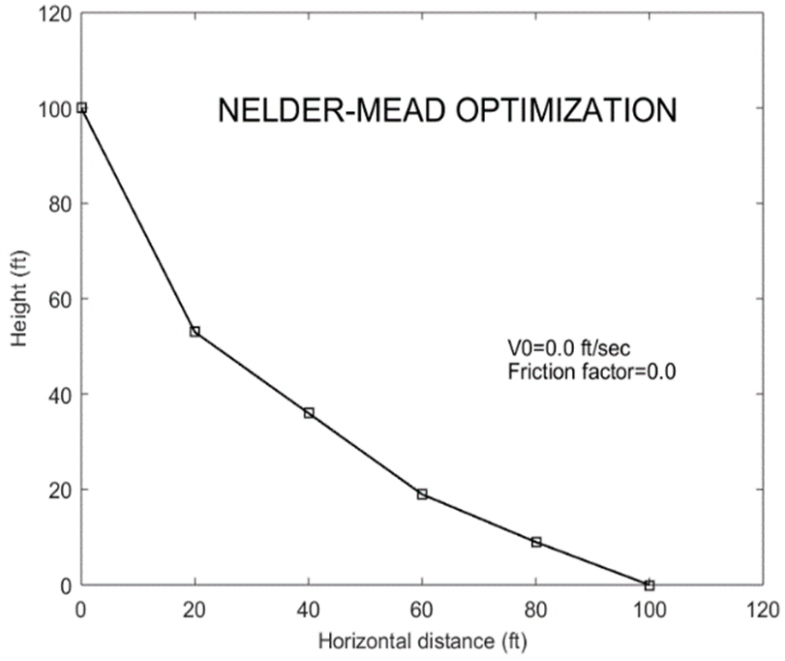

FIGURE 3. Optimal support locations found by Nelder-Mead

Figure 4 shows a direct comparison between the Nelder-Mead (classical numerical method) and the GA optimization for identical initial velocity of 20 ft/sec and 0.5 friction factor. The population size is 200, and the maximum number of generations is 5000 (a high value to discover the best the GA can give). As shown, the two paths are almost identical. The minimum time is 3.39 sec and 3.387 sec by using the GA and the Nelder-Mead respectively. To access GA efficiently, there should be an optimum population size that leads to convergence quickly and effectively.

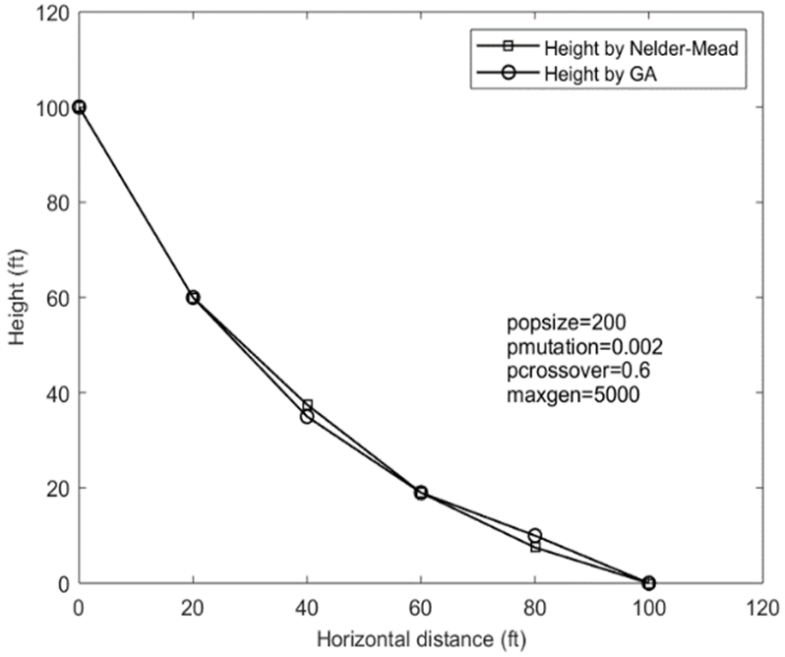

FIGURE 4. Optimal support locations; comparison between Nelder-Mead and the GA

For a small population size, the genetic algorithm will converge too quickly with insufficient processing of too few schemata. On the other hand, population with too many members results in long waiting times for a significant improvement [8,16]. During our study, we are able to obtain good results even for popsizes (i.e., population sizes) as low as 25 and 50. Choosing popsize 25 meant that we have to increase maximum generation size to give GA more chance to explore and exchange more information. Goldberg suggests popsize of 535 for any 40 bit string [10]. We obtained good results for 535 popsize but there is a waste of time and effort since 200 or even 100 popsizes are good enough to get the required optimum solution. Using a popsize equal to 50 means at most 50 suggested paths between the two points. Since the string length is 40 (i.e., 10 bits for each support location), this means every path could have (240) values for the support locations heights (i.e., six support locations). The total number of possible paths for each generation = $50 \times (240)$ at most. If we have 2000 generations, then the total number of paths that could be detected during the whole run equals = $50 \times (240) \times 20001.1 \times 107$ trials of different paths during the whole run. If you add the ability of GA to select good paths, exchange information and reproduce other good paths, there is no doubt that GA will reach convergence before 2000 generations. What we should know is that the performance of GA is somewhat problem dependent and there is no way to determine a performance measure which would be satisfactory in all cases. Moreover, optimum initial population size is dependent on the performance measure used, either offline measure (which weighs convergence) or online measure (which weighs internal performance) [8,17].

3. **The Neural Network – GA Control Surface Approximation.** The genetic algorithm is used, as in Section 2, to extract adequate number of optimum descent paths from point A $(0, 0)$ to point B (a, b), this is done with several initial conditions of initial velocity V_o (the velocity here is just a quantity with no direction). The direction of the particle all the way from the start point until the final point is guided by the values of the support locations of every path. The GA extracted paths that are functions of initial

points (x_o, y_o, and V_o), and consist of intermediate support locations (x, y-values). These values are collectively used as inputs to train a feedforward NN using error back propagation algorithm. The target here is the sliding angle value (direction of the particle) [18,19]. The NN used has two layers: an output layer of 1 neuron and hidden layer of 10 neurons. All use bipolar "tansh" based activations. The trained NN is tested continuously with new random training pairs until it has been verified that the NN has captured the Brachistochrone law, please see Figure 5 that summarizes the steps of this process.

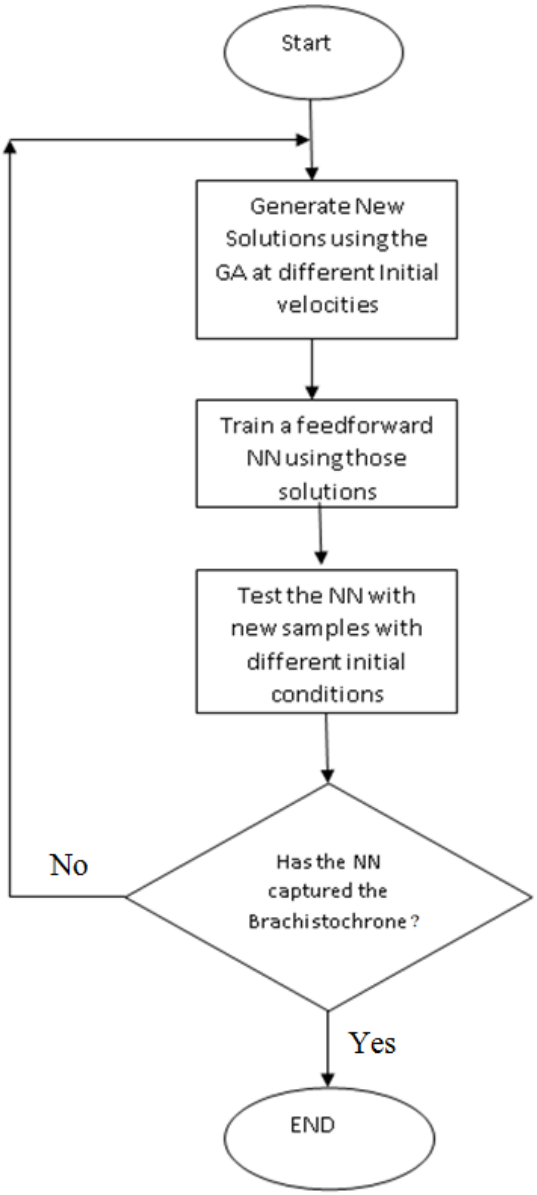

FIGURE 5. Flowchart of training NN with GA solutions

The problem statement, again, is to figure out if the NN can capture the Brachistochrone control. Brachistochrone control means the ability of the NN to generate online decisions regarding the direction of the particle while it is sliding. All solutions that were derived for the Brachistochrone, either analytical or numerical, have the objective of minimizing an integral for the time increment over the path. We are looking, here, to design a compact neural controller that makes online decisions on the particle descending direction within the x-y plane and resulting in minimum time descent.

This controller would have great applications in minimum time path descending applications such as airplanes and jets. The NN is capable of capturing this control surface given adequate set of training pairs extracted from previous solutions to the problem at different initial conditions (GA generated solutions in this case). As mentioned earlier, the input to the NN in this case is the state space vectors of $x(t)$ and $y(t)$ with initial velocity V_o. The output of the NN is the direction: which is the sliding angle: $\tan^{-1}(\Delta y/\Delta x)$, $\Delta x = x(t+1) - x(t)$, and $\Delta y = y(t+1) - y(t)$, t is time step. The training of the NN continues until smooth control surface is achieved and the testing process is verified. Of course the values of $x(t)$ and $y(t)$ are extracted from the optimum paths generated by the previous step.

Training the NN with GA extracted solutions. Around 100 GA generated solutions are used in training a feedforward neural network (NN). A solution is a sequence of 6 points (support locations) in the x-y space, in addition to the initial velocity (V_o) (amount and angle). Each sequence of those 6 points is considered a path (solution). The x and y values of the support locations at every subsequent $\boldsymbol{t}$, in addition to the V_o value, are fed to the NN as inputs, and the calculated sliding angle value ($\tan^{-1}(\Delta y/\Delta x)$) at every $\boldsymbol{t}$ is fed as a target. This is done for each one of those support locations. Bear on mind that the inputs and the targets are based on solutions provided by the GA. If we have 100 extracted paths, then we have 600 pairs of ((x, y) and V_o) (inputs) and sliding angles (targets) for training and testing. The training was done by a two layer NN with 10 hidden units and one output unit. The training was kept continuing until minimal values of errors were reached and the validations were high. The inputs to the NN are fixed V_o, and x, y-locations that are encoded with 10 binary bits and the targets are real radial values of the angle. Any under-learning problem can be solved by reinitializing the weights and adapting the learning rate until the total training error is minimal. Training vectors were divided into 15% for validation, 15% for testing and 70% for learning. The whole process was repeated until high precision and recall is reached. The problem we faced was bad testing results although the learning errors were at the lowest and validation was excellent. This is due to the overlearning problem we had. The NN could perfectly memorize but could not generalize. To solve this problem, we tried to reduce the number of hidden neurons from 10 to 7, and increased the diversity in our training set by mixing it with new solutions for new and different initial conditions. We had to mingle the 600 solutions we have with another 300 and continue training for the already learnt NN with this new mixture. After more than 64000 epochs of training and testing, we managed to reach a compact NN that can generate Brachistochrone solutions for many new initial positions.

4. **Neural Network/Reinforcement Algorithm for Brachistochrone Learning.** Reinforcement learning is totally a different method for training the NN. The NN here is a grid of $(n \times n)$ neurons where every neuron is connected with adjacent left, right, up, and down neurons. Every single neuron has a single weight w_{ij} that is updated according to an input it receives from its adjacent neurons. Please see Figure 6. The collective performance of the neurons in the grid is used to reinforce the neuron that is participating in a proposed path. The neurons that participate more in a proposed path with improvement in the path cost get more rewards than the passive ones which did not participate at all or participated in higher cost paths. The reinforcement is iteratively repeated until a sequence of neurons with distinguished high weights appears within the grid. This sequence should be representing an ancestor of the Brachistochrone solution. The neurons with higher weights have better chance to be selected and be part of a

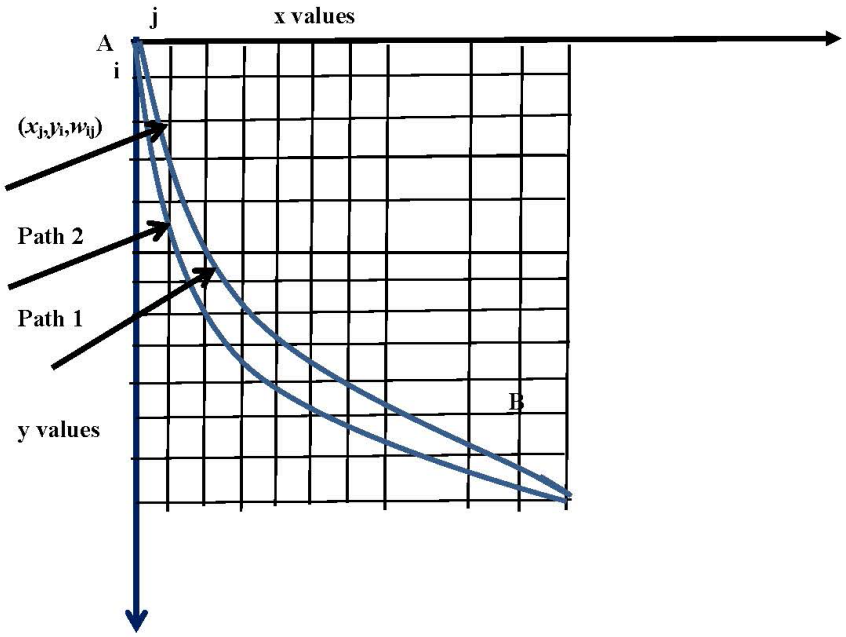

FIGURE 6. The $(n \times n)$ grid path optimization

new proposed path. After enough number of iterations, the highest weights sequence of neurons will form the best path track.

This path will be the solution of the Brachistochrone problem.

The following can summarize the neural network/reinforcement algorithm.

Algorithm Rein-NN:
Initializations
$\boldsymbol{max\text{-}path = 1000}$
Read (n);
Define $i : 1 \rightarrow n$ *(index of y values)*
$j : 1 \rightarrow n$ *(index of x values)*
$k : 1 \rightarrow$ *max-path (index of number of paths)*
Define
$\boldsymbol{w_{ij}}$: *real number variable (weight of point* (x_j, y_i)*)*
x_j: *integer number (1 to n)*
y_i: *integer number (1 to n)*
// resolution could be increased by increasing n.
Define
$\boldsymbol{path\ [k] = [x_j, y_i, w_{ij}]}$ *– from points [i.e.,*
$\boldsymbol{A(0,100)}$ *to* $\boldsymbol{B(100,0)}$*]* // (see Figure 6 please)
Define:
$\boldsymbol{dt_{ij}^k = \left[\left(1 + \left((dy/dx)_{ij}^k\right)^2\right)^{1/2}\right] \Big/ (2 \cdot g \cdot y_{ik})^{1/2}}$

// as time increment (cost) at point (x_i, y_j) within path (k). ($\boldsymbol{g}$ is gravity = 10 //m/sec^2) //
// $(dy/dx)_{ij}^k$ and y_{ik} will be provided for every subsequent point in the path (k) by separate routines that do integration //and interpolation. //
Define
$\boldsymbol{f_k = \sum_{k=1}^{n} dt_{ij}^k}$ (for path (k))
// Start Search: //
Randomize $\boldsymbol{w_{ij}} \rightarrow [0, +5]$ // for the $(n \times n)$ grid (each $\boldsymbol{w_{ij}}$ value is associated // with a x_j and y_i value in the grid) //
// Generate two random paths ($k = 1$ and $k = 2$).
// Calculate f_0 and f_1 (for the two paths); //
Set $\boldsymbol{t = 0}$ // (index of search loop – max = n); //
Do repeat k
Do repeat t
// for every subsequent value of $\boldsymbol{w_{ij}}$ *stored in path* (k) *– where we have* n *values for the whole path update //*
// weights according to learning rule: //
$\boldsymbol{noise = RND[0, 1]}$ *// mutation (minor random value) //*
$\boldsymbol{R_{ij} = |y_{ik-1} - y_{ik}| * C}$ *//calculate reward //*
$\boldsymbol{w_{ij}^k(t+1) = w_{ij}^k(t) + R_{ij} \cdot sign\,(f_{k-1} - f_k) + noise}$
// say $C = 2$. ($\boldsymbol{R_{ij}}$ is the impact that point (x_j, y_i) made to the total cost of the path) //
// Note that y_{ik-1} comes from path $(k-1)$ and y_{ik} comes from path (k), both are associated with the same x_j value //that is identical in path (k) and path $(k-1)$. //
$\boldsymbol{t = t + 1}$
Until $\boldsymbol{t >= n}$ *// finished updating the weights of the points in the path //*
$\boldsymbol{k = k + 1}$ // (new path index) //
For $\boldsymbol{j = 1}$ ***to*** $\boldsymbol{n}$
For $\boldsymbol{i = 1}$ ***to*** $\boldsymbol{n}$
Find $\boldsymbol{(w_{ij})^* = \max(w_{ij})}$ // (done for every j) //
End i
// Stack Path $[k]$ with $\left[x_j^*, y_i^*, (w_{ij})^*\right]$ (note x_j^*, y_i^* are associated with $(w_{ij})^*$) //
End j

// Create a new path with new points and weights, where x_j^* and y_i^* are the point associated originally with $(w_{ij})^*$ being the **maximum weight** for fixed j and variable i (x_j^* and y_i^* values are taken from the $(n \times n)$ whole space). The grid was created initially and then had its weights updated using $w_{ij}^k(t+1)$ equation. //
// ***Now a new path is created.*** //
// (Note we always keep track of two paths path (k) and path $(k-1)$ and ignore the rest) //
// Calculate the new f_k for the new path //
Repeat until k => max-path.
// Optimum path should be stored in path (k). //

5. **Discussions and Analysis of Results.** The simulations are summarized in the table below. The same fixed support locations for $\boldsymbol{x}$ (x-axis values) 0, 20, 40, 60, 80 and 100 were used. The outputs of the search techniques were the $\boldsymbol{y}$ support locations (i.e., optimum path) and the minimum total descent time for the particle. Table 1 shows the average minimum descending time over 100 runs at different initial velocities for the three different techniques explained in the paper (i.e., the GA, GA trained NN, and reinforcement training NN).

TABLE 1. Average times for descending time (100 runs)

V_o (initial speed) ft/s	Average time (s) (GA)	Average time (s) (GA trained NN)	Average time (s) (Reinforcement training NN)
0	3.45	3.56	3.78
5	3.34	3.35	3.57
10	3.33	3.33	3.62
15	3.23	3.20	3.26
18	3.18	3.17	3.25
20	3.01	3.20	3.24

It is clear from Figure 7 and Figure 8 that the GA method had the best solution. Of course the GA works on finding solutions for some specific initial points. The GA does not try to generalize a surface function in a manner similar to what the NN in the other two methods do. The NN as a tool for multidimensional function approximation can interpolate and extrapolate surfaces for this optimum control problem. Hence, the NN is capable of finding "acceptable" solutions for many initial state points that were not part of the training set. However, the GA only provides solutions for given initial conditions without any ability to interpolate nor predict solutions (without search/learning). The GA trained NN through supervised training method is directly given solutions of the Brachistochrone through simple mappings between the $\boldsymbol{x_i}$'s, and $\boldsymbol{y_i}$'s from a side (input) and the sliding angle from the other side (target). Thousands of training epochs for the NN using error-back propagation algorithm result in weaving a control surface that resembles a surface of $(\boldsymbol{x_i}, \boldsymbol{y_i})$ Brachistochrone solutions. The solutions generalize a control law for this benchmark problem. On the other hand, reinforcement learning in the (Rein-NN algorithm) is much more challenging since there is no direct supervision rule to train the NN. The reward/penalty algorithm used could end up with approximate solutions and develop control rules for finding the Brachistochrone at different initial conditions.

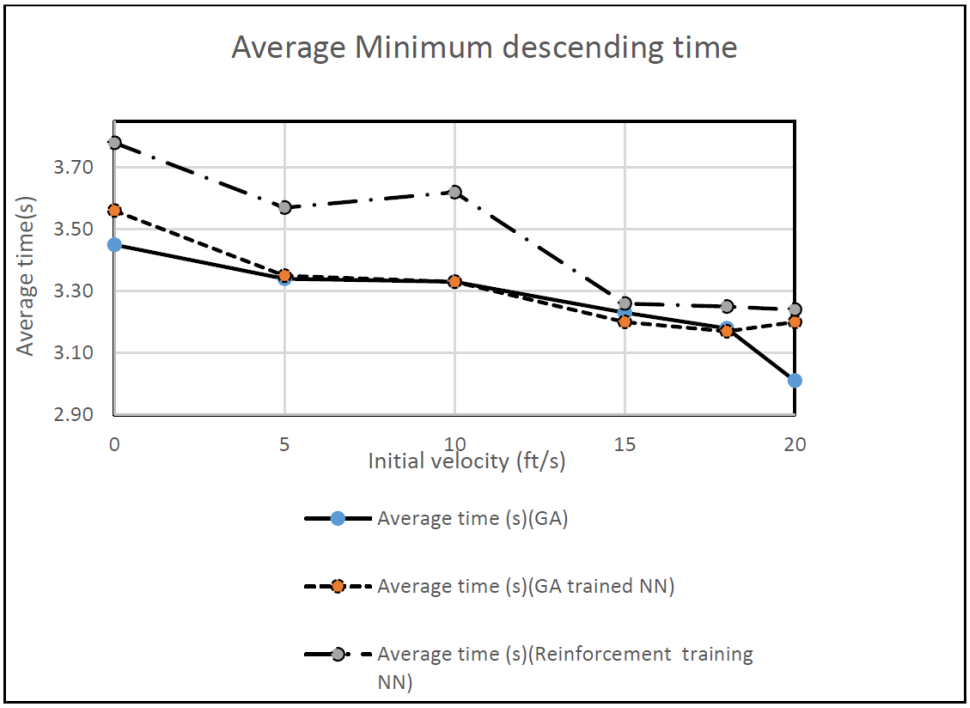

FIGURE 7. Average minimum descending time for the 3 methods

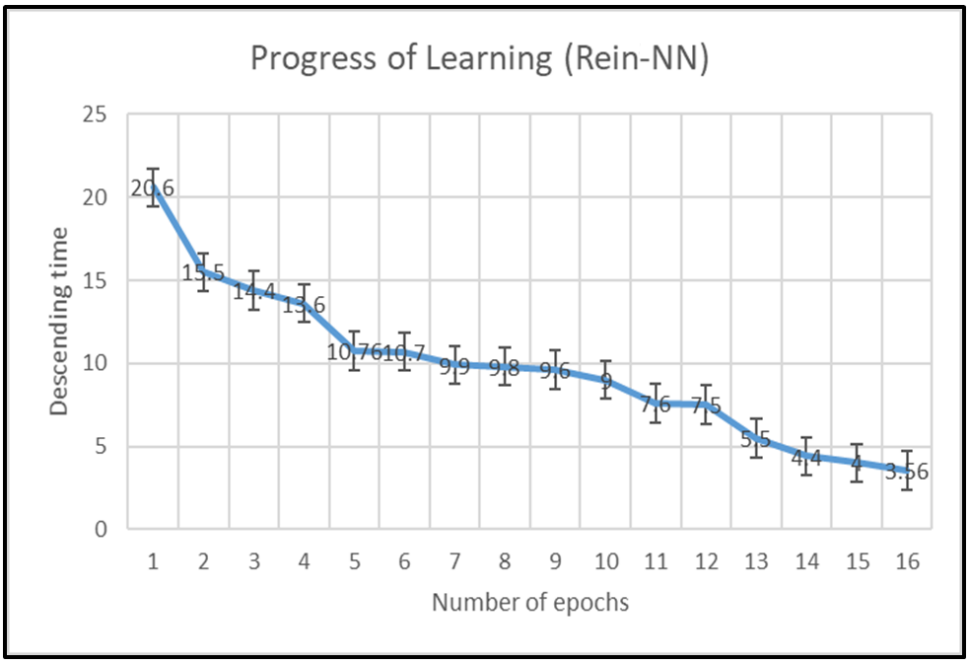

FIGURE 8. Progress of learning in the Rein-NN method for a single initial condition

However, the success rate here was less than that in the GA or even the GA trained NN. Learning through reinforcement is considered one of the most challenging types of learning due to the few pay-off information and the lack of priori knowledge used (no existing control rules).

Table 2 summarizes the percentage of success rate of generating a Brachistochrone at different initializations. A successful attempt here means finding an optimum path from an initial condition that is radius% far as Euclidian distance from some initial condition

TABLE 2. Success rate of testing based on radius distance from training initial conditions

Vicinity (radius% of distance from initial support location – used in training)	GA (success rate)	GA-NN (success rate)	Rein-NN (success rate)
5%	**15%**	**87%**	**62%**
15%	**5%**	**80%**	**59%**
20%	**2%**	**78%**	**55%**

used in training. The fact that GA lacks the ability to generalize was behind this low success percentage. GA is designed to extract single solutions at some given initial points. The GA-NN performance was the best since it is more capable of building a continuous surface that resembles a control law for the Brachistochrone. However, as the radius gets larger and the initializations tend to become at a farther distance from the original support locations (training points), the percentage of successful paths will become less and less. The Rein-NN model comes in between the GA and the GA-NN. Its ability to generalize is better than the GA; however, its outcomes are less than the GA-NN. For us, this success of reinforcement learning means much more than a percentage of successful paths. It is a success for a very different approach of machine learning that is very similar to living creatures' ability to learn by themselves [19,20]. If you notice there were no training sets for the reinforcement method nor there was a supervised learning method. The surface is built through continuous adjustments for the neurons weights in the $(n \times n)$ grid. The "winner takes all" for the column of neurons during training was the method of selection to upgrade the weights of neurons that reside on the path with minimum time (the Brachistochrone).

To summarize the discussion, we re-used the GA and Nelder-Mead algorithms to find solutions for the Brachistochrone problem given initial conditions, we have introduced two new methods for finding solutions for the Brachistochrone problem. We have shown that NN has the abilities to capture solutions for difficult optimum control problems. These are difficult problems since they have many variables, nonlinear functions, and work under constraints. The two NN methods are very different in concept. While the first one is a supervised learning method that uses an already existing knowledge of location pairs as support pillars for building the control surface, the other type is merely a reward/penalty reinforcement learning process working on "paving" threads of paths to weave a control surface that have acceptable performance when tested under non-training initial conditions. The fact that friction or non-friction of the particle and the surface affects stability of solutions can be a study of future work. Techniques similar to that are used in [21-24] should be investigated.

6. **Conclusions and Future Work.** The flexibility of NN and its high adaptability and nonlinearity enabled it to learn an optimum path control problem with acceptable success rates. We do not deny the optimum solutions that the GA or Nelder-Mead (up to six variables) can find for some fixed and given initial conditions; however, the ability of NN to capture and learn by itself and even discover solutions with acceptable success rates is impressive. The minimum pay-off information needed by the reinforcement method and the re-use capability of the GA-trained NN is another beautiful characteristic that those neural systems have. The goal of the paper was not only finding the minimum time of the Brachistochrone, but we could prove that NN can either "memorize then generalize" an optimum control law as we have done with the GA trained NN or it can

even learn a complex control strategy based on simple reward and penalty actions. Future work may include more challenging optimum control problems with more variables and constraints. New training algorithms that utilize the parallelism existing in the $n \times n$ neural grids should be exploited. Parallel programing languages are more suitable to simulate NN algorithms, new types of GA should be tried such as structured-population human community based genetic algorithm (HCBGA) and the cellular genetic algorithm (CGA) and Ising genetic algorithm [13,25]. Those new GAs proved to be more efficient and more effective in solving optimization problems and they could provide better pillars for any control surface that can be weaved by NN.

REFERENCES

[1] *http://www-history.mcs.st-andrews.ac.uk/HistTopics/Brachistochrone.html#s1*, 2002.

[2] P. J. Olver, *Introduction to the Calculus of Variations*, University of Minnesota, http://www-users.math.umn.edu/∼olver/ln_/cv.pdf, 2016.

[3] R. Coleman, *A Detailed Analysis of the Brachistochrone Problem*, https://hal.archives-ouvertes.fr/hal-00446767v2, 2012.

[4] H. J. Sussmann and J. C. Willems, The Brachistochrone problem and modern control theory, in *Contemporary Trends in Nonlinear Geometric Control Theory and Its Applications*, A. Anzaldo-Meneses, F. Monroy-Pérez, B. Bonnard and J. P. Gauthier (eds.), World Scientific Publishing, 2002.

[5] R. A. Zitar, *The Genetic Algorithm as an Alternative Method for Optimizing the Brachistochrone Problem*, Master Thesis, North Carolina A&T State University, Greensboro, 1989.

[6] B. Julstrom, Evolutionary algorithms for two problems from the calculus of variations, *Genetic and Evolutionary Computation – GECC*, Chicago, IL, USA, 2003.

[7] E. Balas and A. Ho, Set covering algorithms using cutting planes, heuristics, and sub-gradient optimization: A computational study, *Mathematical Programming*, vol.12, pp.37-60, 1980.

[8] R. A. Zitar and A. M. Al-Fahed Nuseirat, Performance evaluation of genetic algorithms and evolutionary programming in optimization and machine learning, *Cybernetics and Systems: An International Journal*, vol.33, no.3, pp.203-223, 2002.

[9] J. H. Holland, *Adaption in Natural and Artificial System*, The University of Michigan Press, Ann Arbor, 1975.

[10] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company, Massachusetts, 1989.

[11] R. A. Zitar and E. Hanandeh, Genetic algorithm with neighbor solution approach for traveling salesman problem, *International Journal of Neural Network World*, 2007.

[12] R. A. Zitar and A. Nuseirat, A rule based machine learning approach to the nonlinear multi-fingered robot gripper problem, *Control and Cybernetics*, vol.34, no.2, pp.553-574, 2005.

[13] R. A. Zitar, The Ising genetic algorithm with Gibbs distribution sampling: Application to FIR filter design, *Applied Soft Computing*, vol.8, no.2, pp.1085-1092, 2008.

[14] J.-H. Park and Y.-K. Choi, Optimal control using neural networks for Brachistochrone problem, *Journal of the Korea Institute of Information and Communication Engineering*, vol.18, no.4, pp.818-824, 2014.

[15] R. L. Gonz'alez, *Neural Networks for Variational Problems in Engineering*, Ph.D. Thesis, Department of Computer Languages and Systems, Technical University of Catalonia, 2008.

[16] M. J. Al-Muhammed and R. A. Zitar, Probability-directed random search algorithm for unconstrained optimization problem, *Journal of Applied Soft Computing*, vol.71, pp.165-182, 2018.

[17] R. A. Zitar and A. K. Al-Jabali, Towards general neural network model for glucose/insulin in diabetics-II, *Informatica: An International Journal of Computing and Informatics*, vol.29, 2005.

[18] A. F. Nuseirat and R. A. Zitar, Trajectory path planning using hybrid reinforcement and back propagation through time training, *International Journal of Cybernetics and Systems*, vol.34, no.8, 2003.

[19] R. A. Zitar and M. H. Hassoun, Neurocontrollers trained with rule extracted by a genetic assisted reinforcement learning system, *IEEE Trans. Neural Networks*, vol.6, no.4, pp.859-879, 1995.

[20] M. M. Al-Tahrawi and R. A. Zitar, Polynomial networks versus other techniques in text categorization, *International Journal of Pattern Recognition and Artificial Intelligence*, vol.22, no.2, pp.295-322, 2008.

[21] R. A. Zitar and A. Hamdan, Spam detection using genetic based artificial immune system: A review and a model, *Artificial Intelligence Review*, 2011.

[22] R. A. Zitar, Optimum gripper using ant colony intelligence, *Industrial Robot Journal*, vol.23, no.1, 2004.

[23] R. A. Zitar and A. M. Al-Fahed Nuseirat, A theoretical approach of an intelligent robot gripper to grasp polygon shaped object, *International Journal of Intelligent and Robotic Systems*, vol.31, pp.397-422, 2001.

[24] A. M. Al-Fahed Nuseirat and R. A. Zitar, A neural network approach to firm grip in the presence of small slips, *International Journal of Robotic Systems*, vol.18, no.6, pp.305-315, 2001.

[25] N. A. Al-Madi, K. A. Maria, E. A. Maria and M. A. AL-Madi, A structured-population human community based genetic algorithm (HCBGA) in a comparison with both the standard genetic algorithm (SGA) and the cellular genetic algorithm (CGA), *ICIC Express Letters*, vol.12, no.12, pp.1267-1275, 2018.