1-1-2010

Improved learning in grid-to-grid neural network via clustering

William E. White
University of Memphis, wewhite@memphis.edu

Khan M. Iftekharuddin
The University of Memphis, iftekhar@memphis.edu

Abdesselam Bouzerdoum
University of Wollongong, bouzer@uow.edu.au

Improved learning in grid-to-grid neural network via clustering

Abstract

The maze traversal problem involves finding the shortest distance to the goal from any position in a maze. Such maze solving problems have been an interesting challenge in computational intelligence. Previous work has shown that grid-to-grid neural networks such as the cellular simultaneous recurrent neural network (CSRN) can effectively solve simple maze traversing problems better than other iterative algorithms such as the feedforward multi layer perceptron (MLP). In this work, we investigate improved learning for the CSRN maze solving problem by exploiting relevant information about the maze. We cluster parts of the maze using relevant state information and show an improvement in learning performance. We also study the effect of the number of clusters on the learning rate for the maze solving problem. Furthermore, we investigate a few code optimization techniques to improve the run time efficiency. The outcome of this research may have direct implication in rapid search and recovery, disaster planning and autonomous navigation among others.

Keywords

via, improved, clustering, grid, learning, neural, network

Disciplines

Physical Sciences and Mathematics

Publication Details

White, W. E., Iftekharuddin, K. M. & Bouzerdoum, A. (2010). Improved learning in grid-to-grid neural network via clustering. WCCI 2010: IEEE World Congress on Computational Intelligence (pp. 2318-2324). USA: IEEE.

Improved Learning in Grid-to-Grid Neural Network via Clustering

W. White, K. Iftekharuddin and A. Bouzerdoum

Abstract—**The maze traversal problem involves finding the shortest distance to the goal from any position in a maze. Such maze solving problems have been an interesting challenge in computational intelligence. Previous work has shown that grid-to-grid neural networks such as the cellular simultaneous recurrent neural network (CSRN) can effectively solve simple maze traversing problems better than other iterative algorithms such as the feedforward multi layer perceptron (MLP). In this work, we investigate improved learning for the CSRN maze solving problem by exploiting relevant information about the maze. We cluster parts of the maze using relevant state information and show an improvement in learning performance. We also study the effect of the number of clusters on the learning rate for the maze solving problem. Furthermore, we investigate a few code optimization techniques to improve the run time efficiency. The outcome of this research may have direct implication in rapid search and recovery, disaster planning and autonomous navigation among others.**

I. Introduction

Maze navigation is an important subject for autonomous robots and route optimization. A possible application may be finding the optimized route in an urban area during a disaster. Teaching the robot to navigate through an unknown environment and find the optimal route is a very difficult task. A simplified version of this problem can be simulated by using a random 2D synthetic maze. In this research, we seek to improve the performance of an existing grid-to-grid neural network, namely the cellular simultaneous recurrent neural network (CSRN), for the 2D maze navigation problem by clustering the states of the maze. The CSRN belongs to the family of simultaneous recurrent neural (SRN) networks. The SRN can be used for static functional mapping, similarly to the MLP. It is suggested that the SRN is a more powerful function mapper than the feedforward multi layer perceptron (MLP) [1][4]. The reason for such suggestion goes back to the biological roots of neural networks. The brain contains recurrent links which make it dynamic. In designing an SRN, one can choose any feed-forward network or any differentiable nonlinear function. The mapping function simply describes which network is used. For example, the mapping function in SRN can be just an MLP with an instantaneous feedback loop wrapped around it [4]. A cellular structure can accompany the usual SRN resulting into a Cellular SRN (CSRN). A cellular structure means that the network is made up of a set of cells each consisting of a set of neurons.

The 2D maze representation and traversal problem has already been explored in several works [1][2][3][4][5][6]. The problem has been shown to have promising results using a CSRN in [1], [2], and [3]. In this work, we attempt to improve the learning rate for the CSRN based maze traversing network by clustering relevant information about the maze cells. Our work is motivated by the success of using weighted clustering for improving dynamic learning in a classification problem [7]. However, since the problem domain of maze traversal is inherently different from that of the classification in [7], we study a novel and meaningful approach to clustering for a maze. Our literature search yields a relevant work in which Manor et al. [8] have used clustering to improve the learning in a Q learning problem. Their clustering approach organizes different rooms into groups for efficient traversing. We further experiment with the effect of the number of clusters on learning for our maze traversing problem. Finally, we study a few code optimization techniques for improving run time efficiency. We discuss background information in Section II and present the methods in section III. Results and discussions are presented in section IV while section V presents the conclusion of this work.

II. Background Review

A. Maze Background

For our work, we use a maze that is represented as a square matrix. Figure 1 shows an illustration of the type of maze. Within this square matrix, each cell is classified as an obstacle (black), pathway (white), or the goal (red circle). The original CSRN [5] solves a maze by calculating the number of steps to the goal from any cell along the pathway. The network does not solve the exact number of steps towards the goal, but rather solves a maze in such a way that from any pathway cell the nearest neighbor with the lowest value will point in the direction of the shortest path to the goal. Therefore, from any particular path cell, continually moving in the direction of the neighbor with the lowest value should trace out the shortest path to the goal. With this type of maze, we proceed to cluster in a way that imitates clustering of rooms. Figure 2 shows an example of a maze with rooms in which each cluster approximately represents one of the rooms. We use this notion of rooms as motivation for our clustering of a maze (described in section III A). In

This work was supported in part by National Science Foundation grants MemphiSTEM NSF-DUE 0756738, CSEMS NSF-DUE 0422097, S-STEM NSF-DUE 0630899, ECCS-0738519 and ECCS-0715116.

W. White was with the Intelligent Systems and Image Processing (ISIP) lab at Department of Electrical and Computer Engineering at The University of Memphis, TN 38152 USA. (email: wew106@gmail.com).

K. Iftekharuddin is with the ISIP lab at Department of Electrical and Computer Engineering at The University of Memphis, TN 38152 USA. (e-mail: iftekhar@memphis.edu).

A. Bouzerdoum is with the School of Electrical, Computer, and Telecommunications Engineering, University of Wollongong, Wollongong, NSW 2522, Australia, (e-mail: bouzer@uow.edu.au).

Figure 2 the small arrows point in the direction that leads to the goal. The small "S" in the top left represents the starting position and the "G" in the bottom right represents the goal.

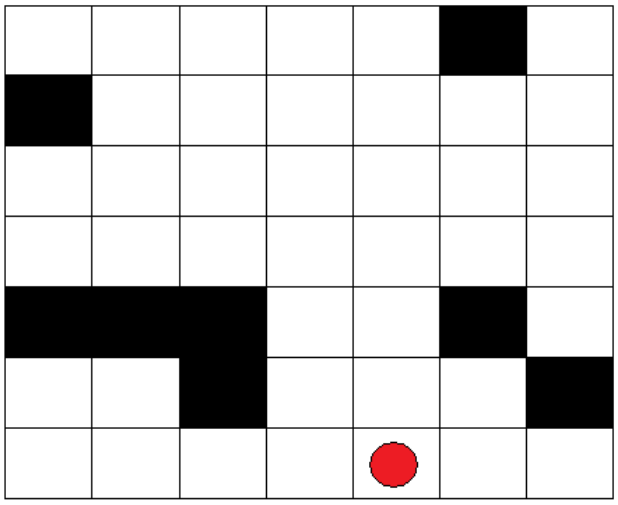

Figure 1: 2D Maze Navigation Problem

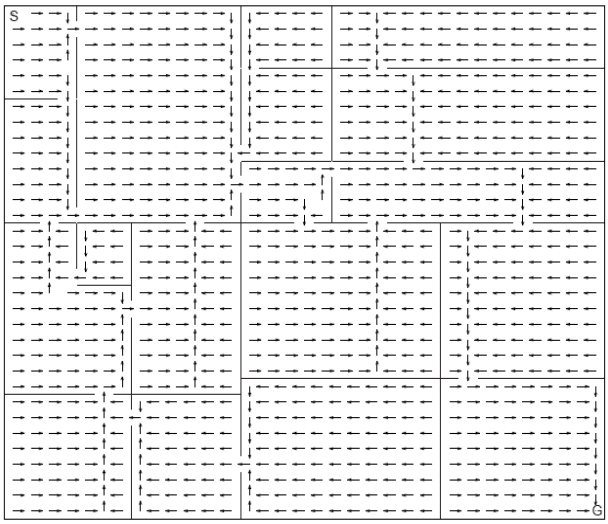

Figure 2: Example maze for room clustering [8]

B. CSRN Background

The CSRN is a combination of a CNN and a SRN. The idea of the CSRN is biologically motivated. The behavior of the CSRN imitates the cortex of the brain which consists of columns similar to each other. The CSRN has been successfully trained with back propagation through time (BPTT). However, BPTT is very slow. In [1], the extended Kalman Filter (EKF) is implemented to train the network by state estimation. The structure of the network is shown in Figure 3. The network is used to process a single cell, rather than the entire maze. A single network is used for each cell in every maze. This drastically reduces the number of required weights. The CSRN is structured so that there are two external input nodes (obstacle and goal), four neighbor nodes, and five recurrent nodes. The obstacle and target nodes use binary values for input. So, any obstacle cell will have a 1 in the obstacle node and a zero in the target node. The target cell will have a 1 in the target node and a 0 in the obstacle node. Therefore, a pathway cell is designated by a 0 in both the obstacle and target nodes. The four neighbor nodes consist of the values of the four nearest cells from the previous iteration. The five recurrent nodes contain the values of the second layer nodes from the previous iteration. This is graphically depicted in Figure 3 as lines from the second layer nodes wrap around to the recurrent nodes. The number of recurrent nodes is defined by a variable within the code that is easily changed.

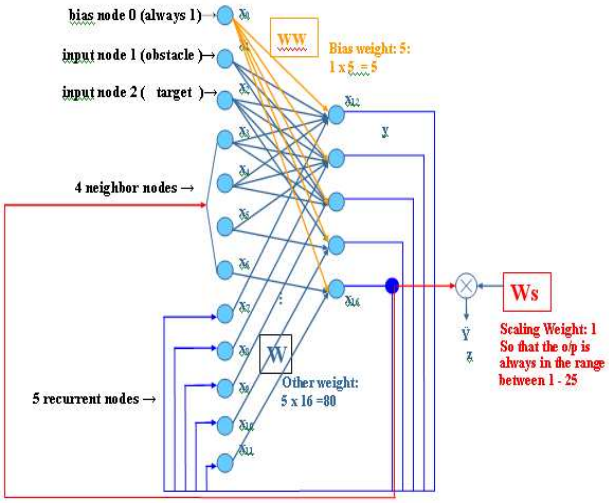

Figure 3: CSRN Network Structure

C. CSRN Training

Since the CSRN is a recurrent network, the Kalman filter works well for its training algorithm since the Kalman filter is a recursive filter that estimates the states of a system based on previous states and current measurements. We briefly summarize the Kalman algorithm below [9]. The following variables are used in the Kalman filter algorithm:

- w(n) is the state vector of the system, the weights in our case
- C(n) is the measurement matrix, or the Jacobian of current output with respect to weights
- G(n) is the Kalman Gain which determines the correction to the state estimation
- Γ(n) is a conversion factor that relates the filtered estimated error to the actual error α(n)
- R(n) is the measurement noise
- K(n, n-1) is the error covariance matrix

The equations can be summarized as follows.

$$\Gamma(n) = C(n)K(n, n-1)C^T(n) + R(n) \quad (1)$$

$$G(n) = K(n, n-1)C^T(n)\Gamma(n)^{-1} \quad (2)$$

$$w(n+1|n) = w(n|n-1) + G(n)\alpha(n) \quad (3)$$

$$K(n+1, n) = K(n, n-1) - G(n)C(n)K(n, n-1) \quad (4)$$

The basic Kalman equations are used to calculate the updates to the weights in the CSRN. These equations also

proved to be a performance bottleneck which is addressed later.

III. Methods

A. Clustering of Mazes

Inspired by the simple maze clustering problem in [8], we attempt to cluster our maze into pseudo-rooms by dividing the mazes into quadrants. We consider the goal in our maze as the reference of origin. Our hopes are that by using this extra information from the resulting clusters, the network will be better able to correctly solve a given maze. Our hypothesis is that one needs to move in the same general direction for cells in each quadrant to reach the goal. However, the goal could be in the corner of the maze which would force the whole maze to group in just one cluster. To eliminate this possibility, we also include the row and column information when clustering. This causes a large quadrant to be split into pieces. One piece of a quadrant might be the half closest to the goal while the second piece might be the half farthest away from the goal. We use k-means clustering with seven variables in our work. The first two variables are the row and column to help split up large quadrants as previously discussed. The next four variables contain directional information (up, down, left and right). The direction is assigned a 1 if moving one cell in that direction moves closer to the goal and a -1 if moving one cell in the direction moves farther away from the goal. The seventh variable contains information about the type of cell. Path cells are assigned a 0 while obstacles are assigned a 1 and the goal is assigned a -1. Once the clusters are defined, we then sort them based on the Euclidean distance between the row and column values of the cluster's centroid and the goal. The steps for our clustering implementations can be seen below.

1. Determine the 7 variables for each cell in the maze
2. Call k-means function in Matlab using 24 clusters
3. Sort clusters based on their centroid's distance to the goal cell

The result of our clustering technique can be seen graphically in Figure 4. Each color represents a different cluster. The dots represent the obstacles. The obstacles all belong to the same cluster, however, cause other clusters to separate. For example, if a cell must travel up and to the right to reach the goal, but there is an obstacle to the right of the cell then the cell might be grouped with a nearby cluster that just has to travel up to reach the goal. To gain a better understanding of Figure 4, note Figure 5 which shows a graphical representation of the solved maze. The darkest red cell towards the bottom in the maze represents the goal and the random dark blue cells are the obstacles. The colors range from red to yellow to blue in order of increasing distance from the goal. In Figure 4, we can see that the cells of a particular cluster must generally travel in the same direction to reach the goal. For example, the cells belonging to the cluster in the bottom left corner must all travel up and to the right in order to reach the goal.

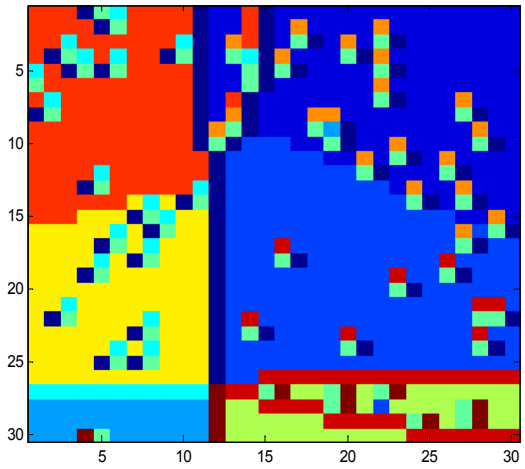

Figure 4: A clustered version of the maze depicted in Figure 5

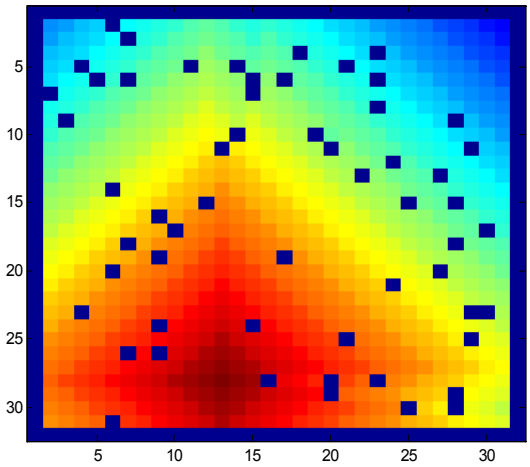

Figure 5: Graphical Representation of a solved maze.

B. Modification of the Existing CSRN

Once we began looking into how to use the clusters we ran into a problem. The original CSRN only had two external input nodes, the obstacle and goal as shown in Figure 3. Therefore, to use the clustered information we would no longer be able to use the obstacle and/or goal as one of the external inputs. We then decided that in order to successfully use any cluster information we would have to be able to use additional external inputs to the network.

Adding additional external inputs proves to be much easier than expected using the built in ability to add extra recurrent nodes using the publicly available CSRN code [12]. We simply redefine how each node is identified. Originally the nodes are identified based on the first node

(the bias node in Figure 3). In this scenario, it may be necessary to redefine several offset pointers embedded deep within the code. However, by redefining the nodes based on the last node (X_{16})in Figure 3, we are able to add additional external inputs by simply increasing the total number of nodes in the network. Therefore, redefining the nodes based on the last one helps one to add new nodes to the front instead of the back.

C. Performance Metrics

We introduce a new performance metric called "Correctness of Solution." This function accurately portrays how well the network has calculated the maze. The correctness of solution considers a cell correct if its neighbor with the lowest value is in the same direction as one of the possible good choices in the target maze. The previous performance metric known as the ""Goodness of Solution", only allows for one "correct" direction. However, Figure 6 shows that a particular cell can have two "correct" directions. The correctness of solution and goodness of solution are both percentages of the number of path cells that point in the correct direction. Figure 7 shows our correctness of solution metric compared with the goodness of solution. Since the correctness of solution considers two possible correct directions while the goodness of solution only considers one, the correctness of solution is always greater than or equal to the goodness of solution as Figure 7 shows. This figure is simply an example of how the correctness is always greater than the goodness and is not meant as a specific example of the performance metric for our solution. However, it is interesting to note that the shapes of the graphs are generally the same.

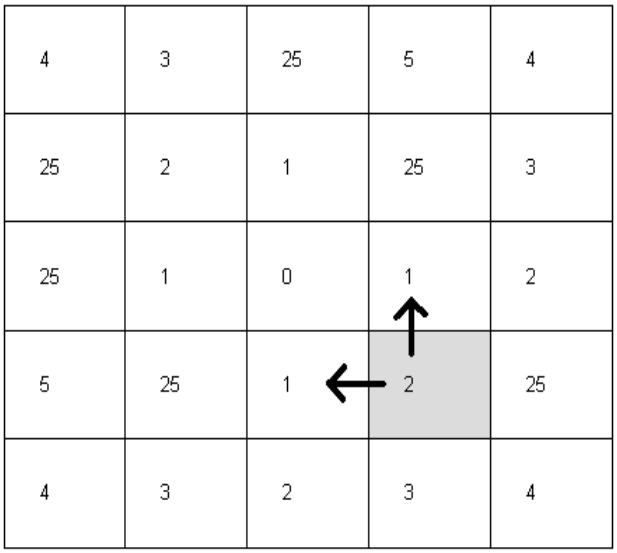

Figure 6: Arrows indicating an example of a cell with two correct directions.

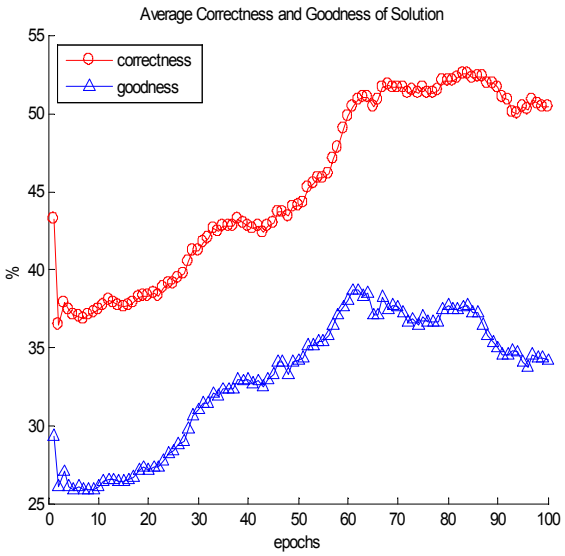

Figure 7: Goodness and correctness vs. epochs.

IV. RESULTS AND DICUSSSIONS

A. Clustering of Maze

We apply our clustering technique to a modified CSRN to obtain improved navigation performance. We also apply the Euclidean distance instead of clusters to the same CSRN and compare our results for maze traversing. These results are obtained using batch of 100 training and testing sets. All training and testing sets used 5 random 12x12 mazes for training and testing. Plots of the correctness of solution versus the number of epochs for the original CSRN and the Euclidean distance method can be seen in Figures 8 and 9 respectively. The original CSRN is not completely stable which can cause a decrease in performance as seen in Figure 8. Similar transient behavior can be seen in Figure 9 but it maintains more steady performance. A condensed summary of our results can be seen in Table 1. The full summary can be found in [6] and includes a method used in [10]. The Euclidean distance technique simply feeds the Euclidean distance of a cell to the goal to the neural network instead of clustering information.

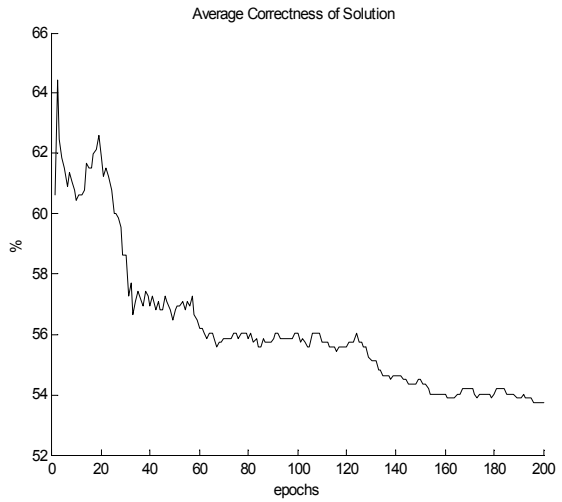

Figure 8: Correctness vs. epochs for the original CSRN (using no additional information)

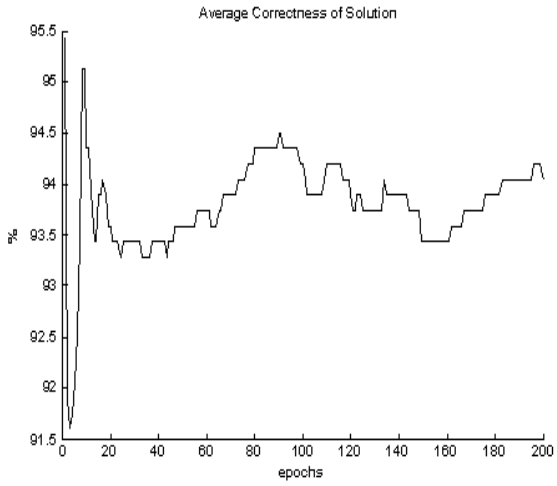

Figure 9: Correctness vs. epochs for the modified CSRN using the Euclidean distance as an external input to the network.

Method	Correctness	Goodness
Original CSRN	67.70%	42.90%
CSRN with Clustering	82.90%	53.20%
CSRN with Euclidean Distance	97.50%	66.20%

Table 1: Comparison of Correctness and Goodness metrics

B. Analysis of the Number of Clusters

The performance of the maze navigation technique should improve as the number of clusters is increased. We also hypothesize that theoretically increasing the number of clusters to the total number of cells in the maze should yield approximately the same performance as does the Euclidean distance technique. To test these hypotheses, we run a batch of 100 training and testing sets. We varied the number of clusters from 4 to 144 using 5 random 12 by 12 mazes. We started with 4 clusters because our initial assumption has been to divide the maze into just 4 quadrants. Our results are plotted in Figure 10.

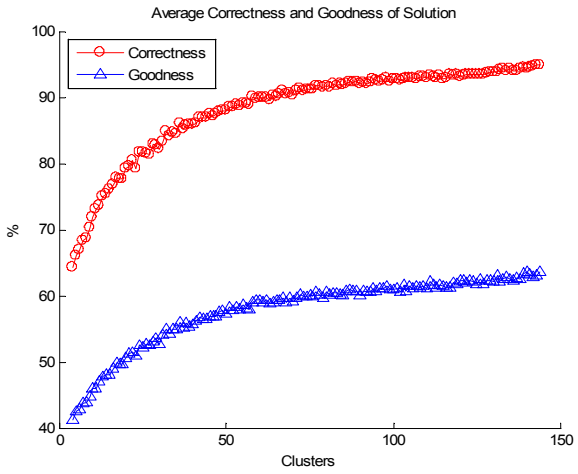

Figure 10: % of performance vs. number of clusters

These results verify our hypothesis that the performance increases as the number of clusters increases. Figure 10 also shows that early increase in the number of clusters yields a greater increase in performance than later increases in the number of clusters. Interestingly, the maximum correctness is 94.78% and occurs with 143 clusters. Following the increasing trend of Figure 10, one may expect the maximum correctness to occur with 144 clusters as we have hypothesized. However, using 144 clusters yields a result of 94.76% which is marginally smaller than that with 143 clusters. Since the correctness value in Figure 10 appears to level off, the small performance difference using either 143 or 144 clusters may be attributed to the the randomization of the mazes.

Although our clustering technique does not exactly match the same performance level achieved by the Euclidean distance, our results suggest effectiveness of our clustering approach in the maze traversing problem. Due to the simplicity of the mazes in this study, it is understandable that the Euclidean distance slightly outperforms our clustering technique. However, the same may not be true for more complicated mazes applicable to real life scenarios. Mazes that contain more walls as opposed to sparse obstacles would not see the same performance.

C. Runtime Performance Improvements

Training the CSRN with the Kalman filter requires taking the inverse of a matrix known as the Kalman gain (see Equation 2). The calculation of the inverse of a matrix is very computationally intensive and dramatically slows down the overall computation of the maze navigation problem.

Therefore, optimization of the existing CSRN code is warranted to produce maze solutions in a timely manner.

A review of code optimization techniques in Matlab [11] shows that instead of taking the inverse of Γ(n), we may simply divide the product of the rest of the terms in G(n) by Γ(n). This simple solution is much more efficient when using Matlab. We applied this solution to the Kalman step function for great results. This change is reflected in Equation (5) which is an optimized version of Equation (2).

$$G(n)=K(n,n-1)C^{T}(n)/\Gamma(n) \qquad (5)$$

Since division is less computationally intensive, this can make a great improvement when working with large mazes and/or many training samples. However, when a few small mazes are used, not much improvement in speed may be expected. Also, it should be noted that multiplying by the inverse and dividing by the matrix do not offer exactly the same result. Due to the rounding of calculations there is a slight difference. For a sample 12x12 Maze the SSE is less than 10^{-3} (1.3231e-004). However, since we are using this matrix for state estimation an exact value is not needed.

We also use the profiler function in Matlab to look for any other bottlenecks and we discovered another bottleneck in the Kalman calculations. When updating the Jacobian matrix, C(n), rows are added one at a time. In Matlab, unlike C, it is easy to add a row to a matrix code-wise. However, even though writing code that augments a matrix is easy, the task is still very computationally intense. When a row is added to a matrix, Matlab automatically creates a new matrix of the appropriate size. Then all the data from the old matrix is copied to the new matrix and the old matrix is deleted. This can also lead to memory fragmentation. In the original code, adding one row at a time became a performance bottleneck in the KalmanAddRowToJacobian function. To overcome this bottleneck, we change the code so that by pre-allocating the number of rows needed we are able to reduce this bottleneck as well.

Our optimization of the original code for the CSRN allowed us to train and test the network much more quickly. As seen in Figure 11 and Figure 12, our first change allows for an improvement from 257 seconds to just 118 seconds in the Kalman step function. For this run, the time for the Kalman step function is decreased by approximately 54%. The second improvement is found in the KalmanAddRowToJacobian function. This improvement decreases the time by approximately 70%. Since these two functions take the most time, the overall time for the main function is decreased from 476 to 218 seconds, a decrease of 54% for the entire program. These results are obtained on a laptop with an AMD Athlon 64 X2 (TK-57) processor with 4 GB of RAM.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
mazev3_main_M5ekf_000	1	476.622 s	1.821 s	
mazev3_KalmanStep	200	257.751 s	257.640 s	
mazev3_KalmanAddRowToJacobian	1000	167.511 s	167.511 s	

Figure 11: Run time for Original Code

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
mazev3_main_M5ekf_000	1	218.497 s	2.296 s	
mazev3_KalmanStep	200	118.316 s	118.237 s	
mazev3_KalmanAddRowToJacobian	1000	48.732 s	48.732 s	

Figure 12: Run Time for Optimized Code

V. Conclustion and Future Works

In this work, we studied the efficacy of clustering in maze traversing problem using a grid-to-grid CSRN. We modified an existing CSRN topology to effectively utilize the maze cluster information. With our modified network we showed that clustering information about a maze can be used to increase the performance of the CSRN. Our study suggests that appropriate clustering of cells in the maze can reach the approximate performance level of using the Euclidean distance information between each cell to the goal to navigate a maze. We further improved the network by making two major changes that reduce the overall run-time by over 50%.

For future work, we plan to analyze the performance for the number of clusters relative to the size of the maze. Also, a study of the computational time due to the varying cluster sizes might add additional insight into the optimal number of clusters. Detailed analysis of the effects of different parameters may also offer a better understanding of what information will help to improve the network's performance. Using more complex and sophisticated clusters may add an interesting new dynamic to the problem. Our maze navigation problem may resembles obstacle avoidance more so than typical maze navigation, but chaining several obstacles together can align the problem space more with what is typically considered maze navigation. Furthermore, one may expand this work of clustering to other application domains such as image processing applications.

References

[1] P. Werbos and X. Pang, "Neural Network Design for J Function Approximation in Dynamic Programming," Math'l Modeling and Scientific Comp., Vol. 2, 1996.

[2] R. Ilin, R. Kozma, and P.J. Werbos, "Cellular SRN Trained by Extended Kalman Filter Shows Promise for ADP," Proc. Int. J. Conf. NN IJCNN 06, pp. 506-510, 2006.

[3] R. Ilin, R. Kozma, P.J. Werbos. "Efficient Learning in Cellular Simultaneous Recurrent Neural Network – The Case of Maze Navigation Problem". IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, 2007. pp. 324-329.

[4] P.J. Werbos and X. Pang, “Generalized Maze Navigation: SRN Critics Solve What Feedforward or Hebbian Cannot,” Proc. Conf. System, Man, Cybernetics, 1996.
[5] R. Ilin, R. Kozma, and P.J. Werbos. "Beyond Feedforward Models trained by Backpropagation: a Practical Training Tool for a More Effcient Universal Approximator" , IEEE Transactions of Neural Networks, June 2008, pp. 929-937.
[6] E. White, Clustering for Improved Learning In Maze Traversal problem, Honors Thesis, The University of Memphis, 2009.
[7] G. H. Nguyen, A. Bouzerdoum, and S.L. Phung, “Efficient Supervised Learning with Reduced Training Exemplars,” IJCNN, June 2008 pp.2981 – 2987.
[8] S. Manor, I Menache, A. Hoze, U Klein. “Dynamic Abstraction in Reinforcement Learning Via Clustering”. ACM International Conference Proceeding Series; Vol. 69. p. 71
[9] S. Haykin, Neural Networks A Comprehensive Foundation, Prentice Hall, Inc., 1999.
[10] K. Anderson, K. Iftekharuddin, E. White and P. Kim, “Binary Image Registration using Cellular Simultaneous Recurrent Networks,” 2009 IEEE Sympsium Series on Computational Intelligence CIMSVP Proceedings, pp. 61-67.
[11] P. Getreuer, Writing Fast Matlab Code, January 2008.
[12] R. Ilin, Publicly available CSRN code, http://cnd.memphis.edu/~rilin/CSRN_Toolbox.html. Accessed on 5/15/2008.