1993

Robotic path planning with obstacle avoidance

Barbara T. Switzer

Rochester Insistite of Technology
Computer Science Department

Robotic Path Planning with Obstacle Avoidance

by
Barbara T. Switzer

A thesis submitted to
The faculty of the Computer Science Department
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by:

Walter A. Wolf

Feredoun Kazemian

Al Biles

August 12, 1993

ABSTRACT

Planning can be used in a variety of applications. In this paper we will discuss those planning techniques that apply to the task of robotic path planning. Here a planner is used to generate "paths" which a robot can follow to maneuver from some point A to another point B, while at the same time avoiding all obstacles.

All approaches discussed in this paper are based on viewing the robot as a sphere. By assuming this, the need to consider the robot's orientation as it moves along a proposed path is eliminated.

Another requirement is that not only must a successful path be found, but this path should also be the shortest path through the space. Since "finding the shortest path between two points that avoids a collection of poly-hedral obstacles in three dimensions is already computationally intractable" [1] and 3-D robotic vision may not be available, the discussion in this paper will be restricted to a 2D plane (this infers that the robot's terrain is a flat hard surface).

Object recognition will also not be considered, only the ability to determine that there is some object present (whether it's a table, chair or T.V. doesn't matter). It's length and width must be known or determined. The height of the object is not important as the robot will go around the

object and not under or over it (can only obtain height information from a 3D plane).

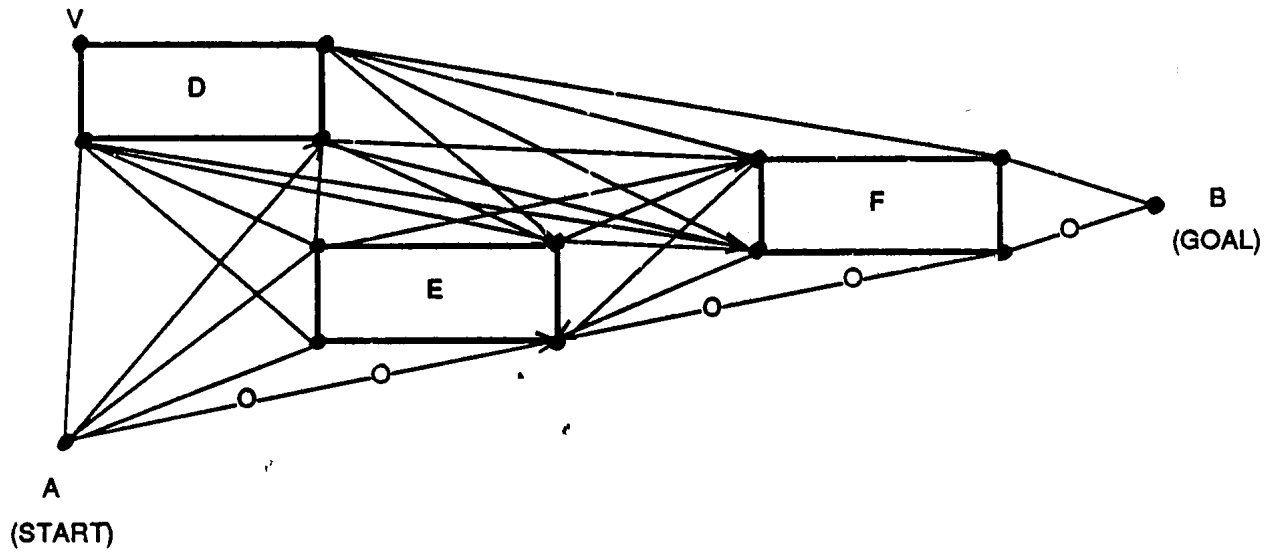
To simplify the overall problem domain we assume that obstacles are not in motion (IE, the objects are not in constant motion; objects can be moved to new stationary locations and new paths around them searched for). The discussion will also restrict the degrees of freedom of the robot to 2. This is again done to reduce the complexity of the domain. As more degrees of freedom are considered, the path planning problem becomes increasingly complex. Finally, we will assume the robot's velocity remains constant (again to reduce the complexity of the domain).

ROBOTIC PATH PLANNING

There are basically two generalized approaches that can be used for robotic path planning; generate and test and graphs. Generate-and-test is the earliest and simplest robotic path planning technique. [2] It consists of a planner which hypothesizes a path, which is then tested by actually issuing commands to the robot to move along the proposed path. If an obstacle is encountered, the robot is moved back to its starting position and the planner hypothesizes an alternate path. This process continues until a successful path is found. Two obvious drawbacks of this approach are that time is wasted by starting the robot back at the start position for each alternate path (extremely wasteful if the robot is only inches away from the goal), and although a successful path is found it may not necessarily be the shortest path. The only real advantage of this approach is its simplicity.

Graphs can be used to assist in resolving the drawbacks of the generate-and-test approach. There are several types of graphs: visibility, configuration space, free space, and state graphs. Visibility graphs are the simplest of the graph approaches. A visibility graph represents the shortest collision-free path between the starting and goal points. The graph is created by connecting all the vertices of objects that are visible to each other (including the start and goal positions) that do not intersect any of the

FIG. 1 - Visibility Graph



KEY:

A = Robot's Starting Position
B = Robot's Goal Position
D = Obstacle
E = Obstacle
F = Obstacle
V = A vertex
O = Robot Moving Along Desired Path

DESCRIPTION:

Lines are drawn from all the vertices of an obstacle to all other vertices of other obstacles (including start/goal positions), with the restriction that the lines cannot intersect the obstacles. This results in all lines representing "collision free" paths.

Notice that no lines are drawn to vertex V, as it is not "visible" to any other vertex or to the start/goal positions.

objects (fig. 1). [3] The lines drawn are guaranteed to be collision free and Euclidean distance can be used to find the shortest path. This approach does not take into account the robot's dimensions, which may lead to the robot "scraping" along the boundary of an obstacle. This problem can be resolved by utilizing a technique known as "growth of obstacles".

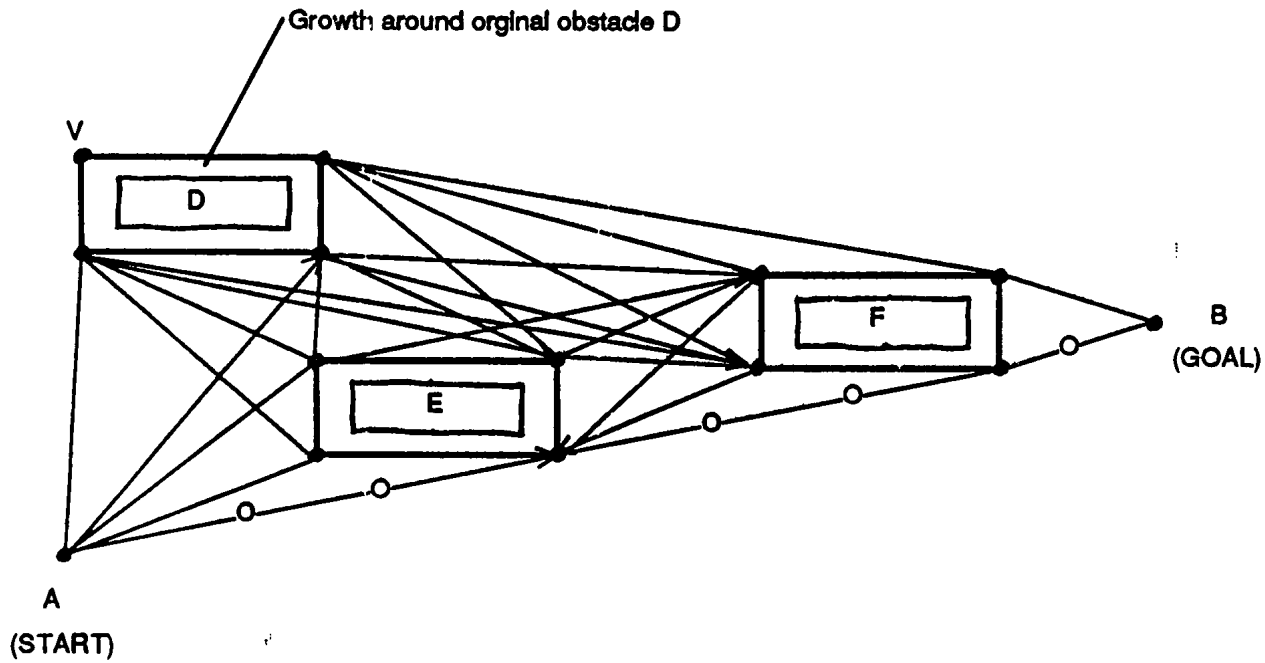
Growth of obstacles is accomplished by adding a constant or varying amount to the actual size of the obstacle (to the obstacle's boundaries, forming new boundaries). The visibility graph is then formed using these new "grown" obstacles. [4] The result is a safety margin around each obstacle, which will prevent the robot from "scraping" along the obstacle's boundaries (assuming the obstacles are grown by a sufficient amount) (fig. 2).

The amount of growth can vary resulting in a variety of safety margins. Then each "growth" is assigned a penalty, where the further away from the obstacle's original boundaries, the higher the penalty. [5] The visibility graph is created as before, but the path may pass through different safety margins of the obstacle (fig. 3). This technique dramatically increases the number of vertices that have to be connected, as this has to be done for each growth of the obstacles. By summing up the penalties along a path one can determine the length of a path (the smaller the penalty the shorter the path).

The use of visibility graphs is limited as it is a local

2;3

FIG. 2 - Visibility Graph with Constant Growth



KEY:

A = Robot's Starting Position
B = Robot's Goal Position
D = Obstacle
E = Obstacle
F = Obstacle
V = A vertex
O = Robot Moving Along Desired Path

DESCRIPTION:

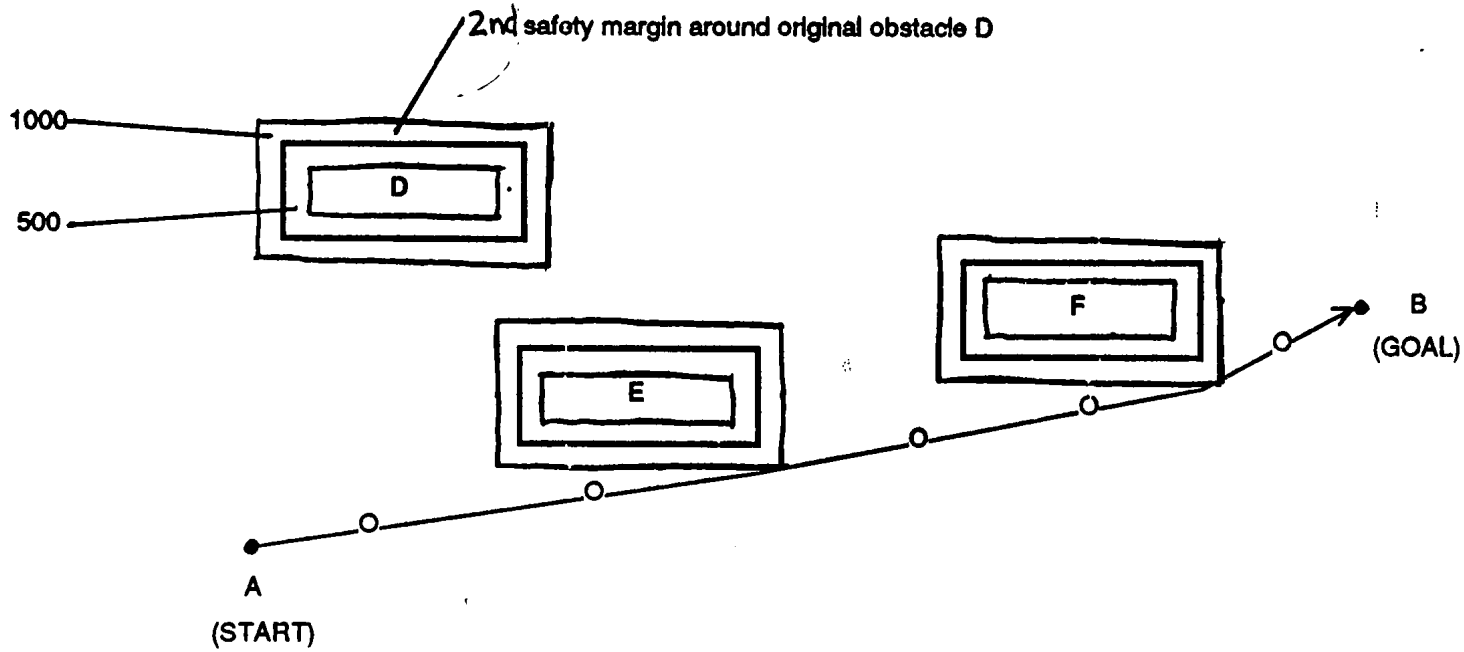
Lines are drawn from all the vertices of an obstacle to all other vertices of other obstacles (including start/goal positions), with the restriction that the lines cannot intersect the obstacles. This results in all lines representing "collision free" paths.

Notice that no lines are drawn to vertex V, as it is not "Visible" to any other vertex or to the start/goal positions.

"Growth" of the obstacles D,E, and F is represented by the distance between the inner and outer rectangles. Growth of obstacles helps prevent the robot from scrapping the obstacles.

3;4

FIG. 3 - Visbility Graph with Varying Growth



KEY:

A = Robot's Starting Position
B = Robot's Goal Position
D = Obstacle
E = Obstacle
F = Obstacle
O = Robot Moving Along Desired Path
1000 = Penalty for Path in 3rd Growth
500 = Penalty for Path in 2nd Growth

DESCRIPTION:

Normally we would draw lines from ALL vertices of ALL growths. We choose not to do so here for diagram simplification.

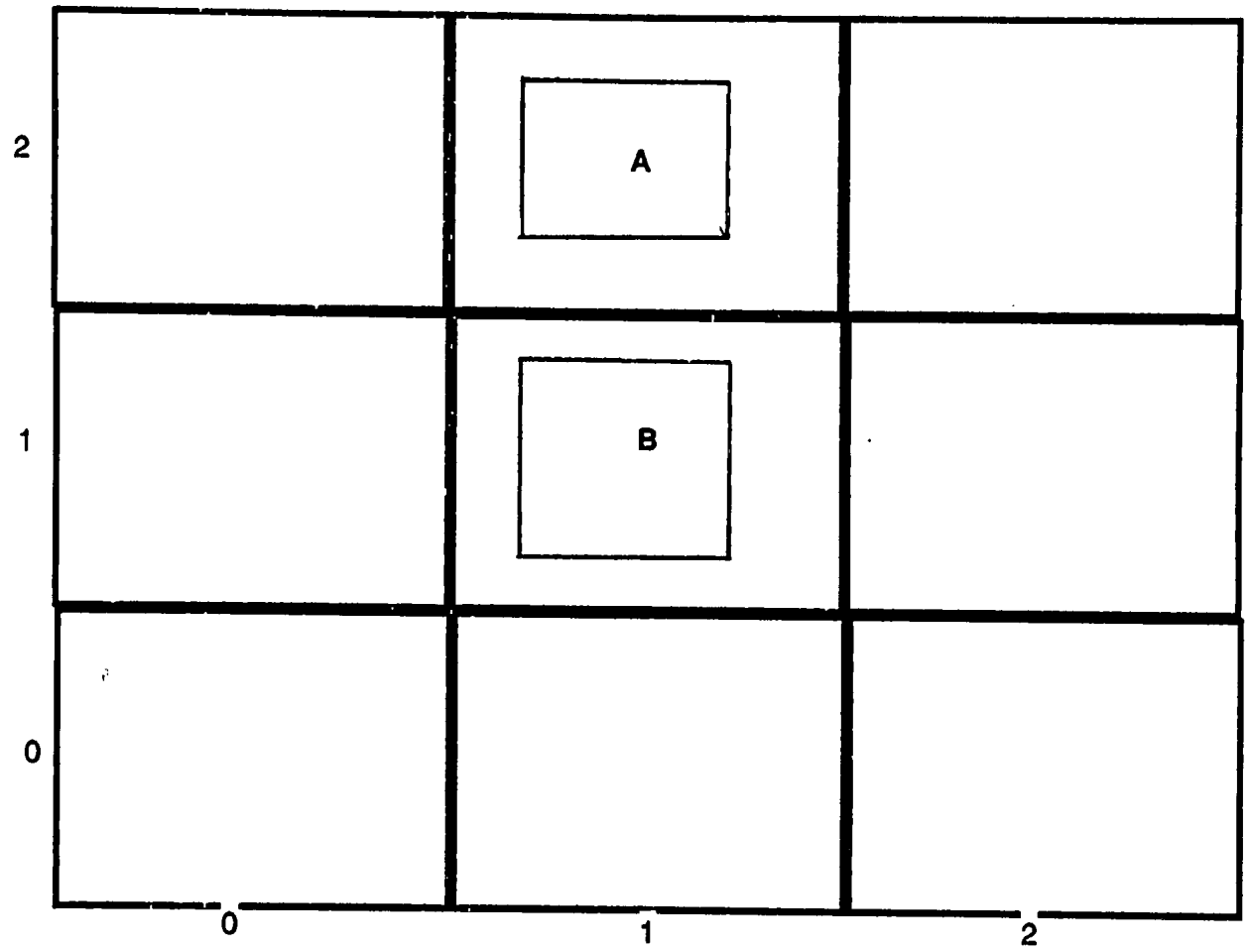
approach (uses only local information). It may lead to cases where no further progress can be made resulting in the robot having to backup as in the case of generate and test, but in this case the robot does not have to move back to the starting point. This approach would be sufficient in cases where only small modifications to the path would be required.

A more robust approach is the use of a configuration space graph, which is a representation of all points in a robot's environment to which the robot can move (the robot's Configuration Space). Graphs of this type are usually constructed by partitioning the robot's space (ex: room, table top, etc.) into fixed size cells (fig. 4). When the robot moves to a particular cell the it is said to be in that configuration. In a 2D plane with 2 degrees of freedom the "configuration" of a cell is it's X,Y coordinate location on the plane. A graph can then be created by making each cell a node in the graph with links connecting adjacent cells (fig. 4 and fig. 5).

While it is possible to find a path using the configuration space graph, one has to take into account the presence of obstacles in some of the cells of the graph. A heuristic would have to be added to the path search to take into account the location(s) of these obstacles. For example, from figure 4 we can see that we do not want to select a path that moves the robot into cells (1,1) or (1,2), as they are occupied by obstacles. The addition of another search method to find the shortest path through the graph

FIG_4;2

FIG. 4 - Grid Representation of a room with Obstacles

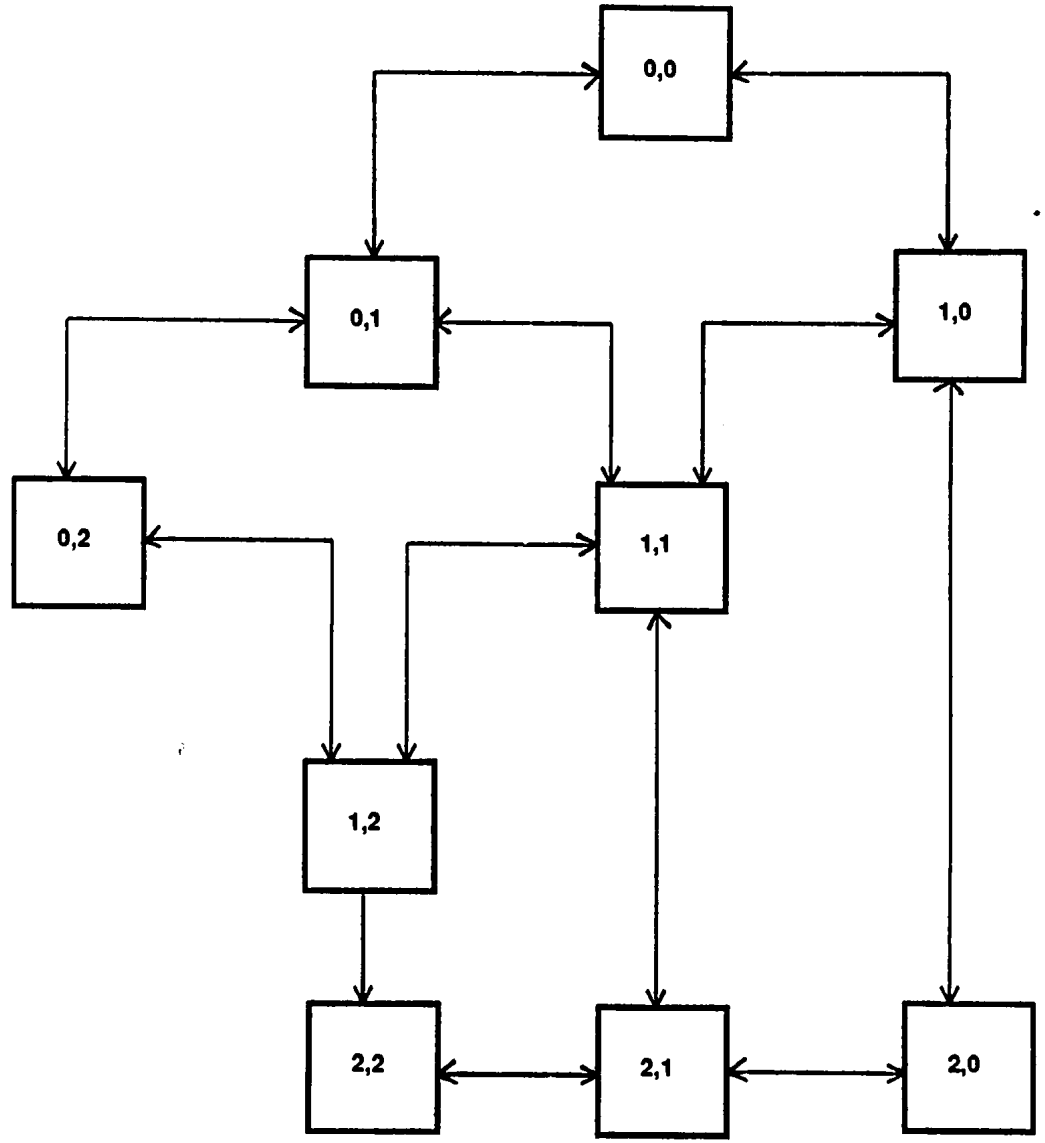


KEY:

A = Obstacle

B = Obstacle

FIG_5;2
FIG. 5 - Configuration Space Graph



Description:

Represents the Robot's Configuration Space based on Fig. 4 - "Grid Representation of a room with Obstacles". Each square on the diagram above corresponds to a cell in the grid shown in Fig,. 4.

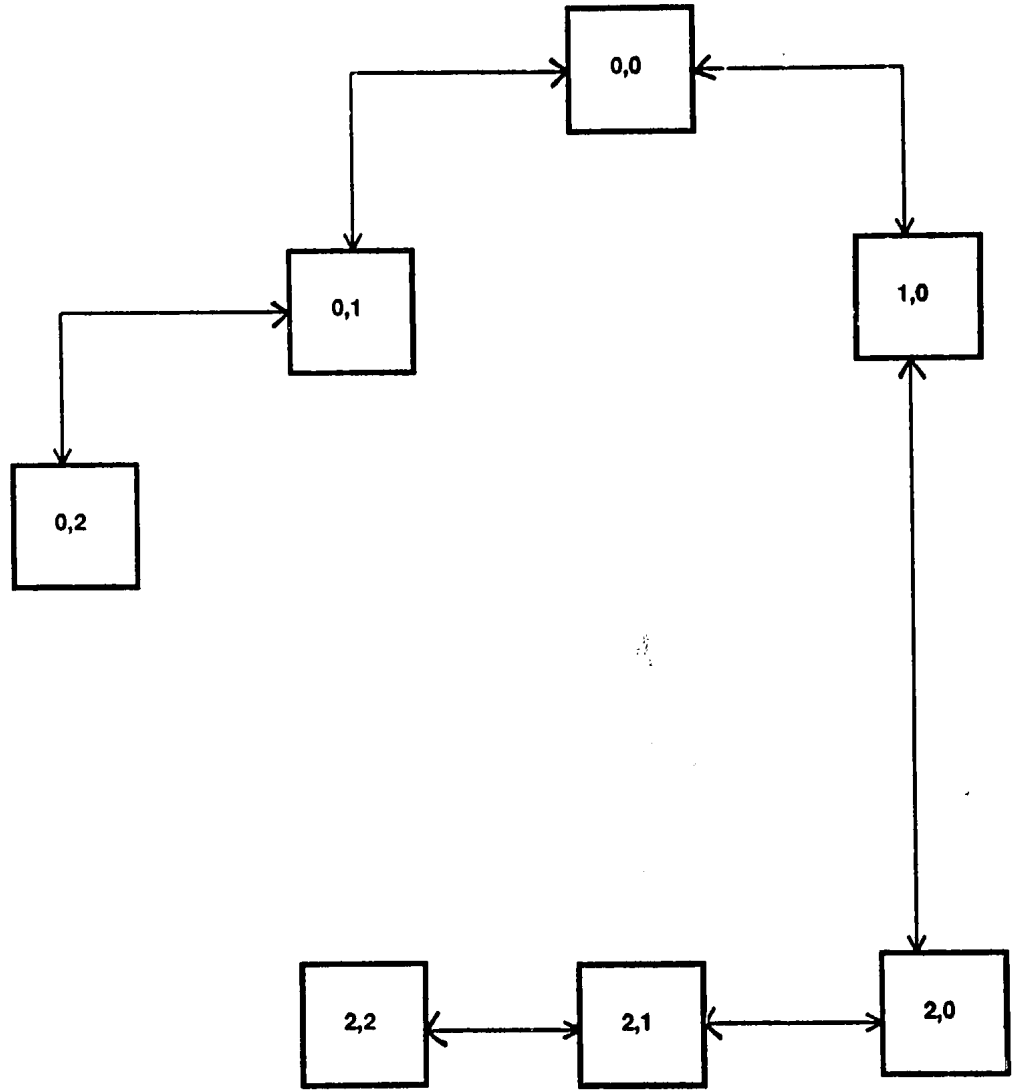
makes this approach even more complex.

To reduce the complexity of the path search, one could use a Free Configuration Space Graph (Freespace Graph). A freespace graph is similar to the configuration space graph with the exception that cells occupied by obstacles are removed from the graph (fig. 4 and fig. 6). The graph then represents only the free configuration space and any path found is guaranteed to be collision free. This simplifies the path search problem and allows more emphasis to be placed on finding the shortest path. [6]

While both the configuration space and freespace graph approaches have their advantages, both may have a problem with allocating enough storage space to store the graphs. This problem arises because fixed size cells approaches "typically require huge numbers of cells to achieve sufficient accuracy." [7] This then results in large search trees, as 1 cell is equal to one node in the graph. This makes the search for paths tedious and time consuming (although the freespace graph search would be faster than that for the configuration space graph). An alternative to these approaches would be to use variable size cells. One such approach has been proposed by Tomas Lozano-Perez (1981) [8].

Lozano-Perez divides cells in the configuration space into 3 types: FULL, MIXED, and EMPTY. FULL cells are completely occupied by an obstacle, EMPTY cells are totally devoid of any object, and MIXED cells consist of some EMPTY

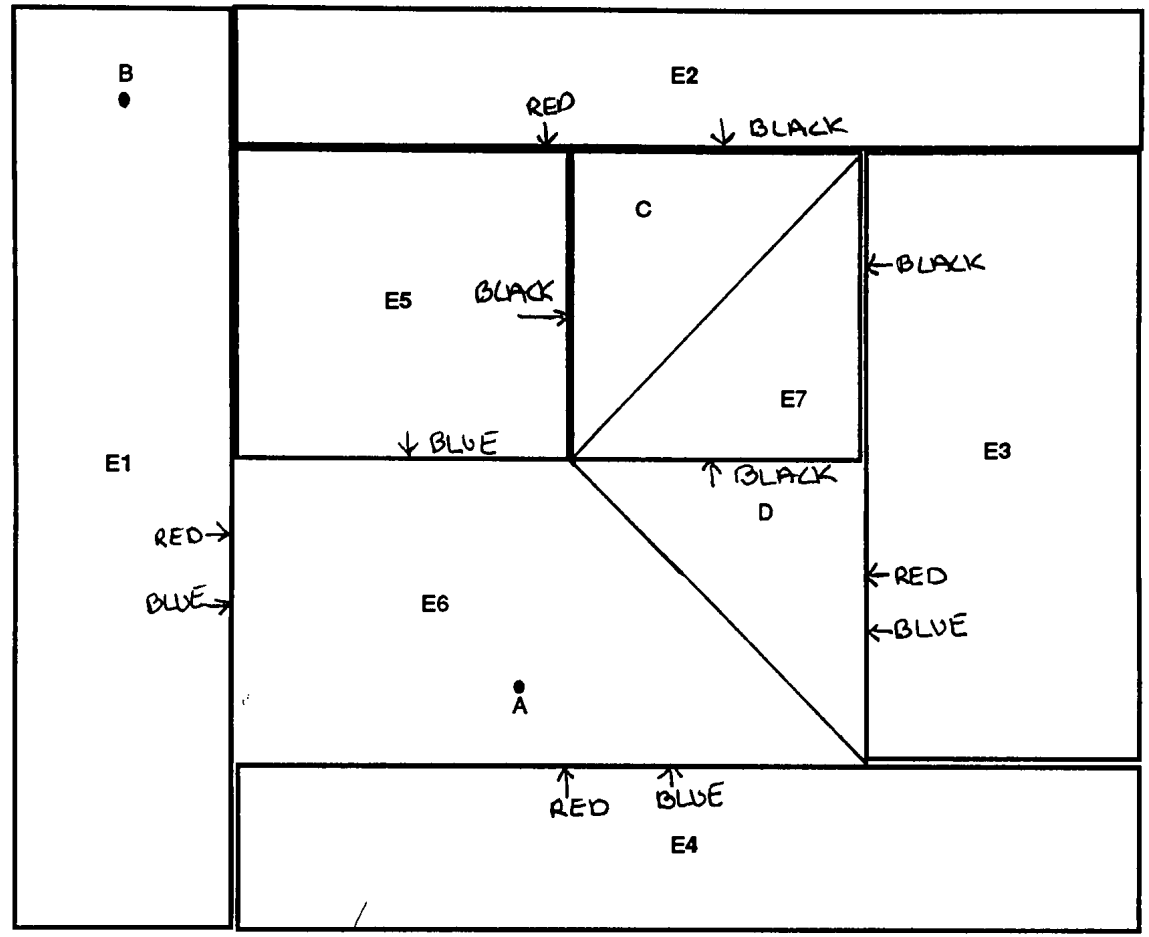
FIG_6;3
FIG. 6 - Freespace Graph



Description:

Represents the Robot's Configuration Space based on Fig. 4 - "Grid Representation of a room with Obstacles". Each square on the diagram above corresponds to a cell in the grid shown in Fig,. 4. This Figure is similar to that shown in Fig. 5 - "Configuration Space" with the removal of Cells [1,1] and [1,2], as they contained obstacles. All cells above represent the Configuration Space "Free" of Obstacles.

FIG_7;4
FIG. 7 - Configuration Space - Varying Cell Size



KEY:

A = Robot's Starting Position
B = Robot's Goal Position
C = Obstacle
D = Obstacle
E1-E7 = Empty Cells (no obstacles in these cells)

RED = Root Cell 1 (refer to Fig. 8)
BLUE = Root Cell 2 (refer to Fig. 8)
BLACK = Root Cell 3 (refer to Fig. 8)

Description:

This diagram is similiar to that in Fig.4 except that the room IS NOT seperated into fixed size cells but VARYING SIZED Cells.

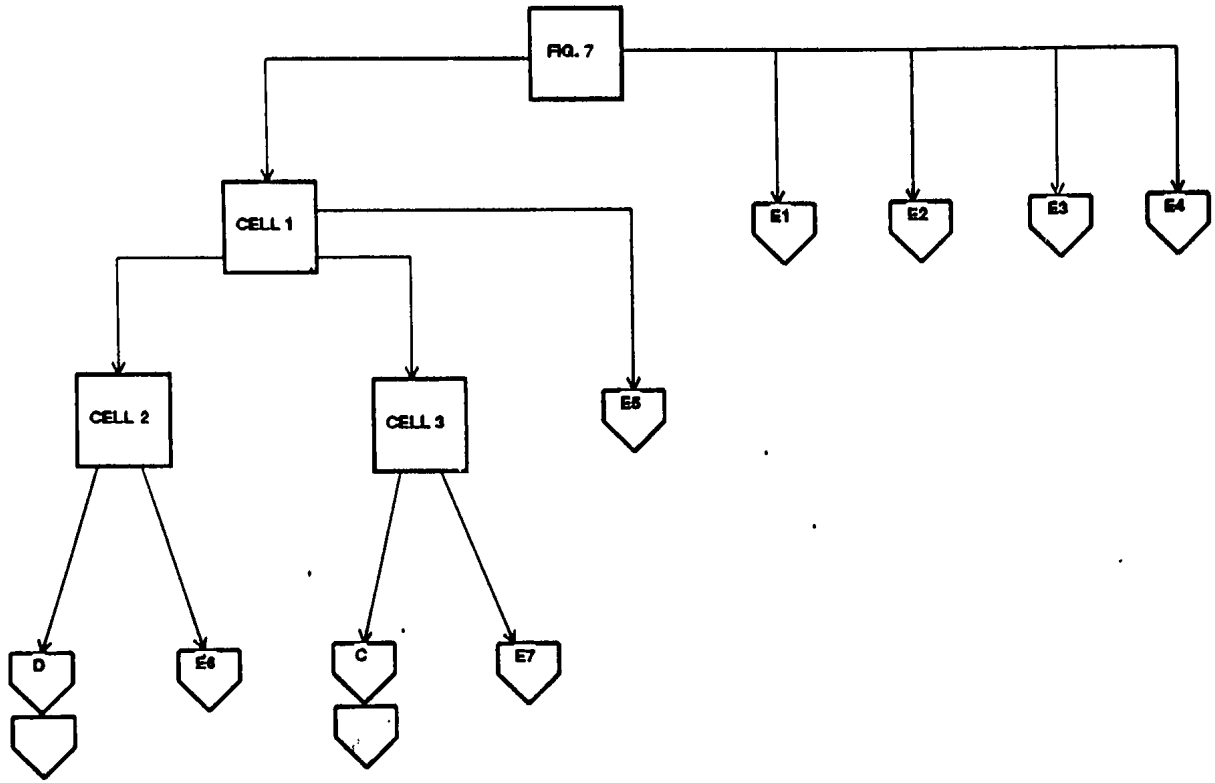
space and some occupied space (fig. 7). The configuration space graph is then created by viewing the boundary of the space as the root cell. This "root" cell then consists of both MIXED and EMPTY cells. The MIXED cells are broken down into their parts until no more MIXED cells are found (fig. 8).

Path selection then proceeds by first, at a global level, making sure that a collision free path is feasible. This is done by using a technique that verifies that the starting and goal positions are in EMPTY cells. If they are not, the path search can be terminated immediately without wasting further processing time (as any paths generated would contain unavoidable collisions). This verification is accomplished by selecting the largest EMPTY or MIXED cell (in that order) that contains the starting location. If the location is in a EMPTY cell the process stops, if not, the MIXED cell is broken into it's components. This process continues until an EMPTY cell with the starting location is found, if not the search fails. This process is then repeated for the goal position.

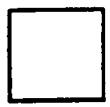
Once EMPTY cells are found for both the starting and goal positions, the freespace graph is then drawn using EMPTY cells as nodes with adjacent cells being linked together (fig. 9). These links can then be weighted to represent distance between the cells to assist in the search for the shortest path.

Although this approach seems feasible, it is unclear as

FIG_8;3
FIG. 8 - Graph of the Configuration Space - Varying Cell Size



KEY:


= MIXED CELL, Contains FULL and EMPTY Cells


= EMPTY CELL, Contains NO OBSTACLES

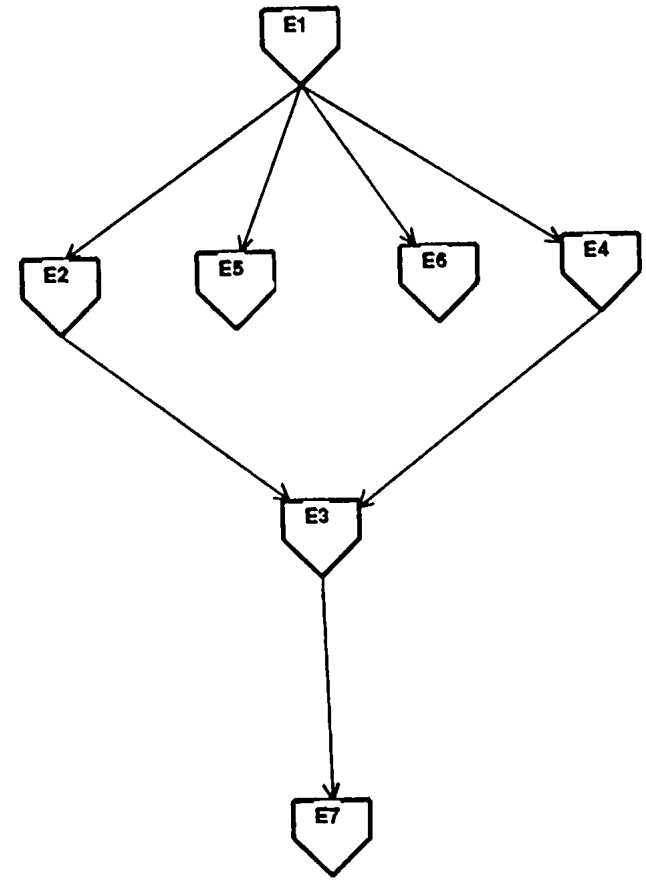

= FULL CELL, Contains an OBSTACLE

Description:

Using the diagram in FIG. 7, the room is graphed using EMPTY, FULL, and MIXED Cells as components of the graph.

Cell 1 was outline in RED in FIG.7, Cell 2 in BLUE and Cell 3 in BLACK.

FIG_9;4
FIG. 9 - Free Configuration Space Graph - Varying Cell Size



KEY:

= EMPTY CELL, Contains NO OBSTACLES

Description:

Using the Graph from Fig.8 we remove all FULL cells. The resulting graph represents all cells that are "OBSTACLE FREE".

to how distance would be calculated so that weights can be placed on the links of the freespace graph. Also, how it is that the sizes of the various cells in the initial graph are determined? If fixed size cells are not used, how would the robot be moved through the space (no mention of coordinate positions)? One last problem is that if the path search fails, it does not necessarily mean that a successful path does not exist (poor approximations of the obstacles).

Another way to view the configuration space is as fixed size cells where each cell represents a "state" and the links between cells are "state transitions". In other words, represent the configuration space as a state transition graph. Faverjon and Tournassoud (1987) [9] have proposed such an approach. When a robot is moving into a cell "state" it is making a "state transition". Once the robot is in that cell it is said to be in the "state". In addition, each "state transition" has associated with it a probability which indicates the probability of making a successful "state transition". The probabilities are initially set to 0.5 and re-calculated using the formula $Pn = s+1/n+2$, where n is the number of events on the transition and s is the number of successful events. The product of these probabilities along a path is then taken as the probability of that path succeeding.

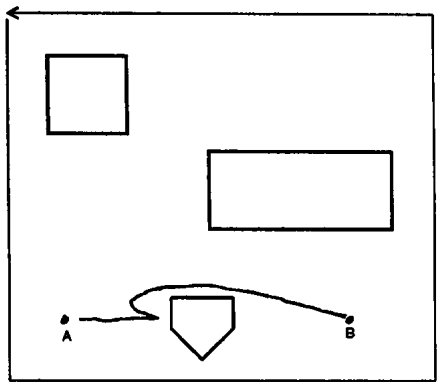
Once the graph is created, a global planner searches for the path with the highest probability of succeeding. This path is then given to the local planner for execution.

The local planner then executes the path by selecting a subgoal in the adjacent state (cell) and trying to move the robot to that position. The subgoal selection is accomplished by selecting, as a subgoal, a point in the middle of an adjacent cell. This first approach results in a "drunkard's walk" [10] (no anticipation as to what direction the robot is moving in). A solution to this problem would be to select a point in an adjacent cell that is both the shortest distance from the current position and the shortest distance to the goal position (fig. 10).

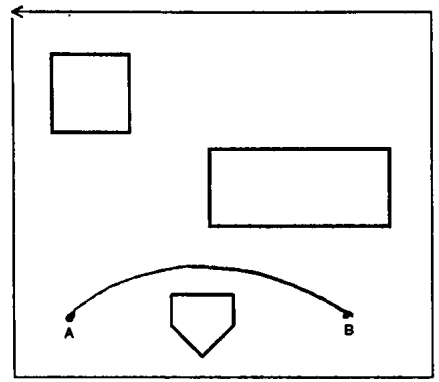
If, as the robot is moved to the subgoal, the robot becomes blocked (hits an obstacle) the corresponding "state transition" probability is decreased by increasing only the number of events on the transition (n). The robot is then moved back into the previous "state" and the global planner is notified. The global planner then generates a new plan, using the updated probabilities (based on the current configuration). If the robot moves successfully to the subgoal, the corresponding "state transition's" probability is increased by incrementing both the number of events on the transition (n) and the number of successful events.

The advantage, of course, is that this approach takes into account local information and may be more appropriate in a world where the ability of sensors to gather accurate global information about the environment is unrealistic. On the other hand, the first time through a space can result in many backups, but because the probabilities of the "state

IG_10.2
IG. 10 - Subgoal Selection



* On the first pass thru the room the robot will have to backup when an obstacle is encountered and go around the obstacle.

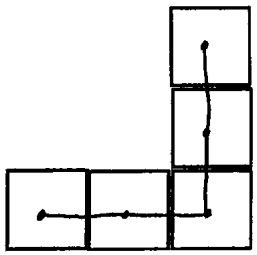


* On the second pass thru the room the robot "learns" from the first pass and navigates around the obstacle.

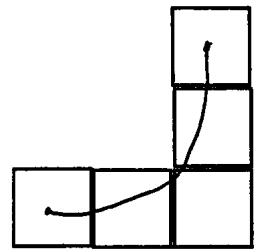
KEY:

A = Robot's Starting Position
B = Robot's Goal Position

FIG. 10



* Subgoal Selection in Middle of Next Cell. "Drunkard's Walk"



* Subgoal Selection in next cell which is the shortest distance from the current and goal positions.

FIG. 11

transitions" are periodically updated, the robot is actually "learning" something about the nature of the space it is moving through. So the second time through the same space from the same starting and goal positions will generate a successful path immediately (fig. 9).

Although all the approaches discussed have their advantages, those that included the use of a global and local planning fit better into real world applications. Due to the limited sensor capability of many robots (plus the limits of today's technology) we cannot be guaranteed that any global information concerning a robot's configuration space is accurate (may be shadows, object behind other objects, etc). In addition, true accurate representation would have to include some application of 3D vision in a 3D plane. It has been mentioned previously that not only does working in 3D add to the complexity of the problem, but finding the shortest path through a 3D environment is not computationally feasible by today's standards. These facts, have lead me to conclude, that the use of both global and local planning is required.

Both the configuration space proposal by Tomas Lozano-Perez [8] and the state graph proposal by Faverjon and Tournassoud [9] utilize some form of a global and local planning (the state graph approach more so). It is a combination of these two approaches, with additional features added, that I will present below as a viable solution to the problem of robotic path planning.

My approach consist of breaking the robot's configuration space into fixed size cells. This allows for easier calculation of distances (aid in finding shortest path) and creates an environment in which issuing commands for robotic movement is simplified. A state graph is then created from the configuration space (as described by Faverjon and Tournassoud [9]). The state graph's advantage is that there exists the possibility of performing some work in parallel (although I will not address this here).

The state graph utilizes probabilities on the "state transitions" success (initially these are set to 0.5). In addition, if a cell has greater than some percentage of it's space occupied by an obstacle, the cell is left out of the state transition graph (this percentage can then be tuned for increased speed). This results in a state graph that represents a pseudo free configuration space graph. This technique of removing some cells results in a successful path being found sooner than by using a straight configuration space approach, and the amount of computation time required to create the graph will be less than that for a true free space graph.

Once the state graph is created, the local planner then attempts to issue commands to the robot to make a state transition into one of the robot's adjacent cells. If the transition fails, the probability of the transition is decreased and another adjacent cell is selected (subgoals selected as middle of adjacent cell for simplicity purposes).

Once a successful state transition is found the process stops and is repeated from the goal position. If no successful state transitions are found, then no successful path is possible and the path search process is terminated. Successful transitions must be found for both the start and goal positions for path searching to continue (similar to Lozano-Perez's [8] approach).

If all is successful at this point, the global planner then generates a path as described by Faverjon and Tournassoud . [9] The intent here, is to maximize the probability of success while minimizing the path distance (number of cells in the path).

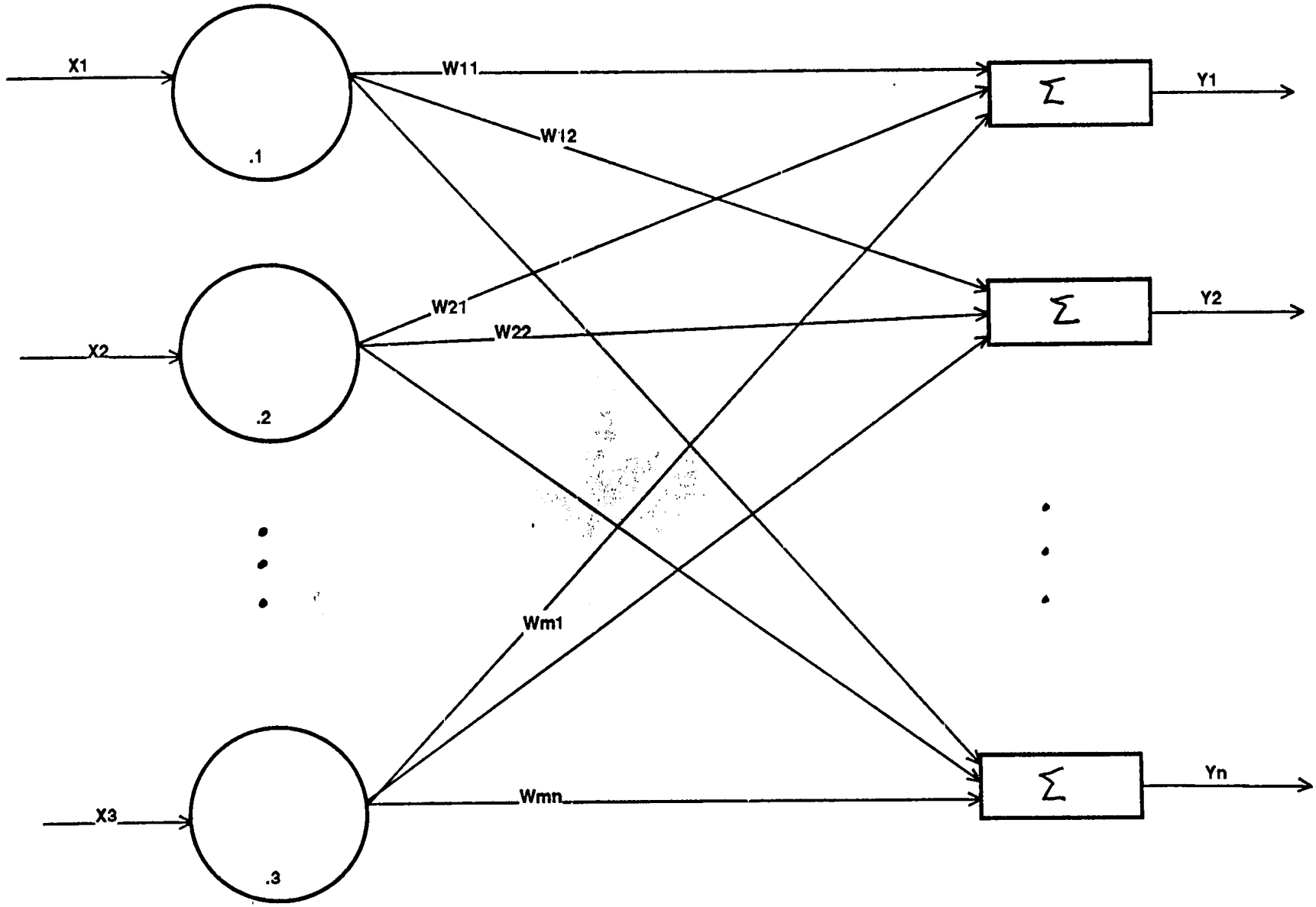
NEURAL NETWORKS

An area of computer science that can be utilized to solve the shortest optimal path problem is that of neural networks. Neural networks attempt to model the thought processes, memory functions and structure of the human brain.

The human brain consists of millions of interconnected neurons, where a neuron is considered to be the basic building block of the human brain and receives/sends signals from/to neighboring neurons. When a signal is received it is averaged with signals from other neurons and, if some predetermined activation threshold is exceeded, the neuron fires, sending a signal onto other neurons. When a neuron is activated, the result of the summation function is sent thru an activation (squashing) function, which determines the output of the neuron (ex: SUM > threshold, OUTPUT = 1 else OUTPUT = 0). Activation functions are usually sigmoidal functions that "squash" or counteract the impact of extremely large signals and increase the significance of very small signals providing needed non-linearity (also known as logistic or squashing functions).

The first simple neural network developed is the perceptron. A perceptron consists of a signal layer of nodes (neurons) connected by weights to a set of inputs (fig. 12) [11]. Each input is multiplied by a weight, all of which are summed and if the result is greater than some predetermined threshold value, the output is 1 otherwise it is 0. In this fashion a neural network consisting of perceptrons can

FIG. 12 - Single Layer Neural Network [11]



KEY:

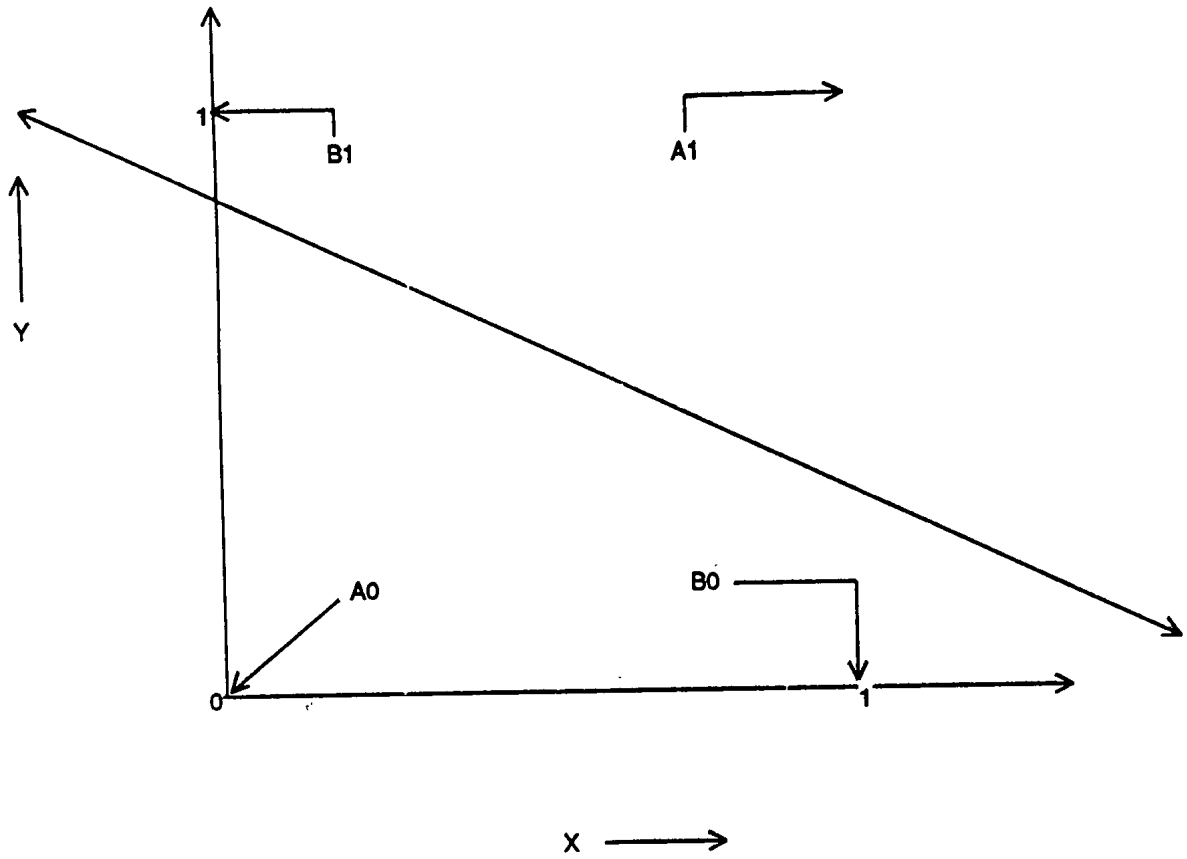
W = Weight
X = Network Input
Y = Network Output

recognize simple patterns by identifying an input as belonging to some class. Minsky found that perceptrons are "seriously limited in it's representational ability; there are many simple machines the perceptron cannot represent no matter how the weights are adjusted". [10]

One example of a machine that the single layered perceptron neural network can not represent is the exclusive or function. The problem stems from the fact that the exclusive or function is linearly inseparable, which means that you cannot separate the input values geometrically (fig. 13) [12]. If two inputs can be separated by a straight line, which places the input's in different output classes, it is linearly separable. When a problem is linearly inseparable (as with the exclusive or function) the inputs are so tightly meshed together that an exact output classification can not be accomplished.

A way to avoid problems with linear separability is to use a multiple layer neural network. Multilayer neural networks are formed by linking single layer neural networks together, where the output of one layer is the input into the next (more closely represents the structure of the human brain). In the single layer network a half plane decision region is created when a straight line is drawn between the inputs (as discussed earlier) (fig. 13) [12]. With a two layer neural network a "convex" decision region is formed by the intersection of the half-plane regions formed by each node in the first layer (region has as many sides as there

3;2
FIG. 13 - Exclusive-Or Problem [12]



Exclusive-Or Truth Table

Point	X Value	Y Value	Desired Output
A0	0	0	0
B0	1	0	1
B1	0	1	1
A1	1	1	0

Description:

In PLANE XY, a half plane decision region is created. The first half contains points A1 and B1, the second half contains points A0 and B0.

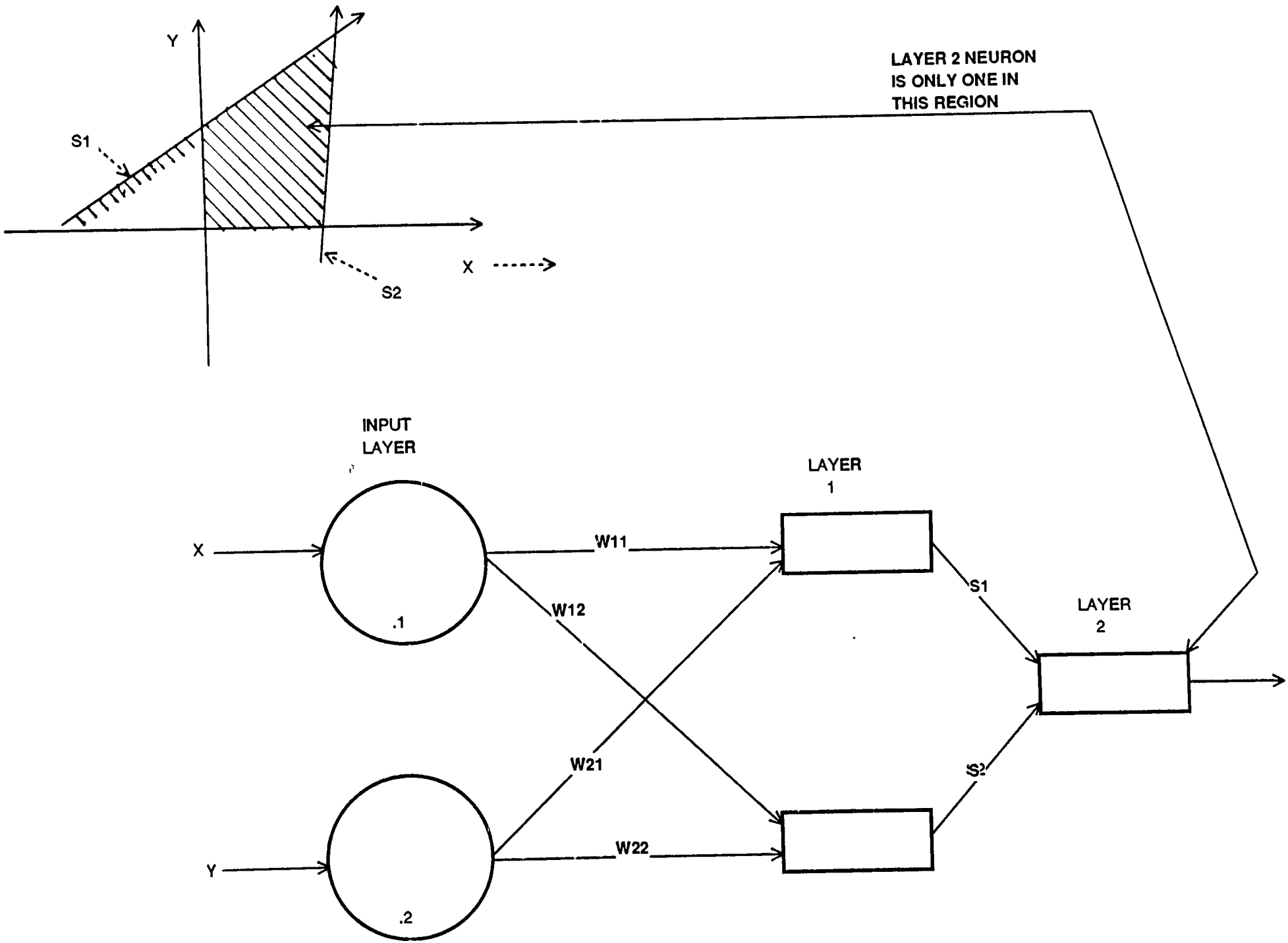
are nodes in the first layer) (fig. 14) [13]. When a third layer is added more complex convex regions can be formed which allow for better separation of tightly meshed classes of inputs.

Once a neural network is created, it must be trained to produce the desired results. The act of training closely resembles the human mind's ability to learn and consists of applying a set of inputs to the neural network then adjusting the weights to achieve the desired outputs.

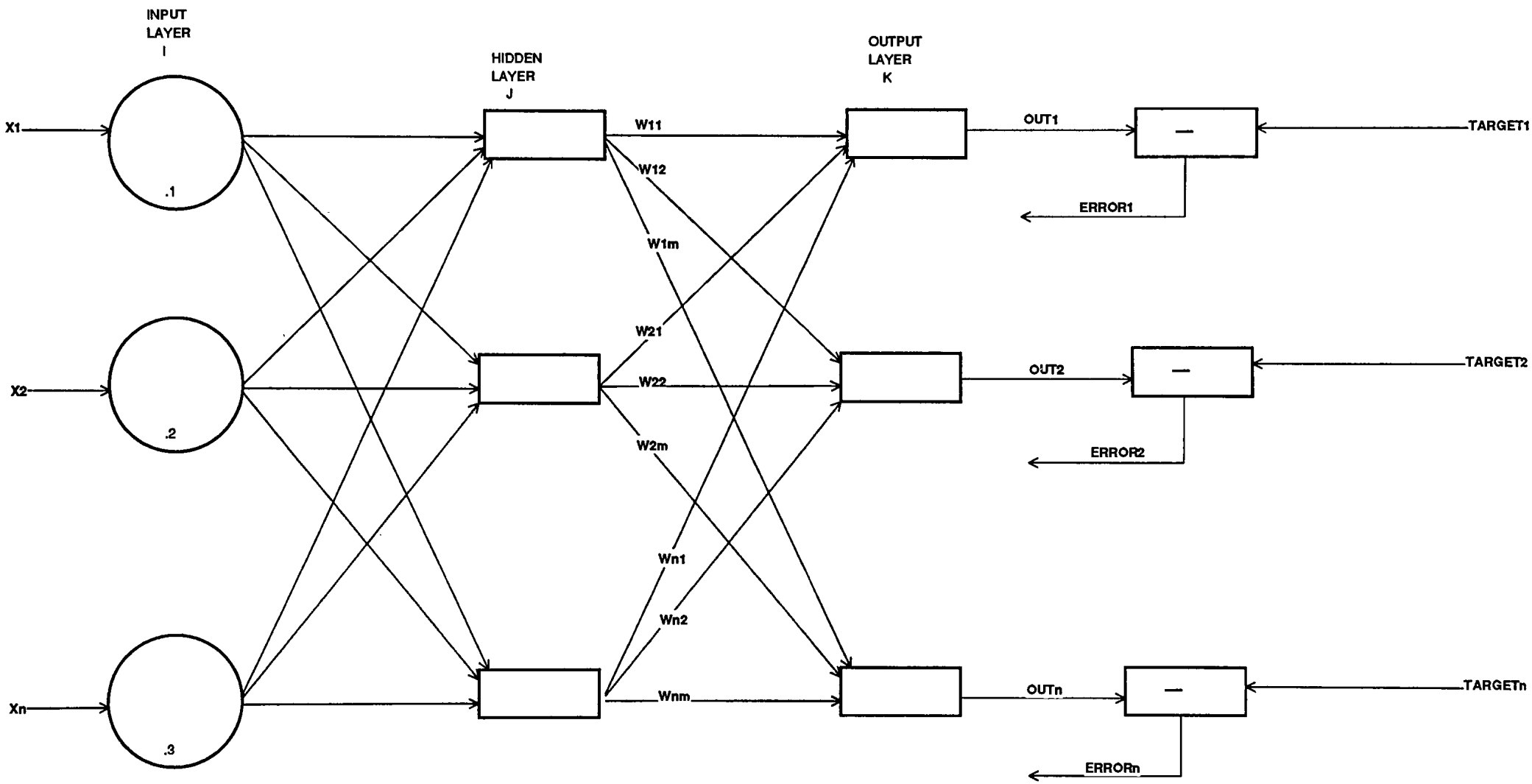
There are two basic categories of training, supervised and unsupervised. Supervised training involves pairing an input with a desired output (training pair). The output of the network is calculated and compared against the desired output. In the case of perceptron training the object is to classify some input (input must match some expected output, ex: trying to recognize the letter A in the alphabet). With other types of supervised training the difference between the desired output and the actual output is calculated and is known as the "error". This "error" is backpropagated thru the neural network adjusting weights to minimize the "error".

The method of supervised learning discussed above is known as backpropagation (fig. 15) [14]. Backpropagation works in the manner described previously for supervised training with the addition of a "squashing function". The "squashing function" is used when calculating the output, to prevent extremely large inputs from dominating the

142
FIG. 14 - Convex Decision Region Produced by Two Layer Net [13]



5;4
IG. 15 - Two Layer Backpropagation Network [14]



calculations (which could lead to erroneous output) and increases the importance of smaller inputs. This has a smoothing effect on the output of the neural network and assists in finding global instead of local minima.

Unsupervised training does not require that the desired output be known in advance this more closely mimics biological learning. A vector of training inputs is applied to the network and the weights are modified to achieve a "consistent" output. Some relationship must be established between the input and output to know when to stop the training process. This could be when the rate of change in outputs slows to some acceptable value at which point one could consider the training to be complete.

An example of unsupervised training is Boltzman training. A variable T (eg. temperature) is set to a very high value and an input vector is applied to the network from which the output is produced by an "objective function". An "objective function" is some function whose output is to be minimized by training the neural network. Once the objective function is calculated random weight changes are performed on the network and the objective function's value is re-calculated. If the new objective function is smaller than the old one, then the weight changes made are retained. If it is larger the weight changes are undone and different random weights are chosen.

When the new objective function's value exceeds that of the old one (this is normally not desired), a Boltzman

distribution is used to determine the probability of accepting the changes. This is done to allow steps in the wrong direction to avoid local minima (fig.16) [15]. A major drawback of this type of unsupervised training is that it can take an inordinate amount of time to train the network. To speed up the network training one could use Cauchy instead of Boltzman training (fig. 17) [16].

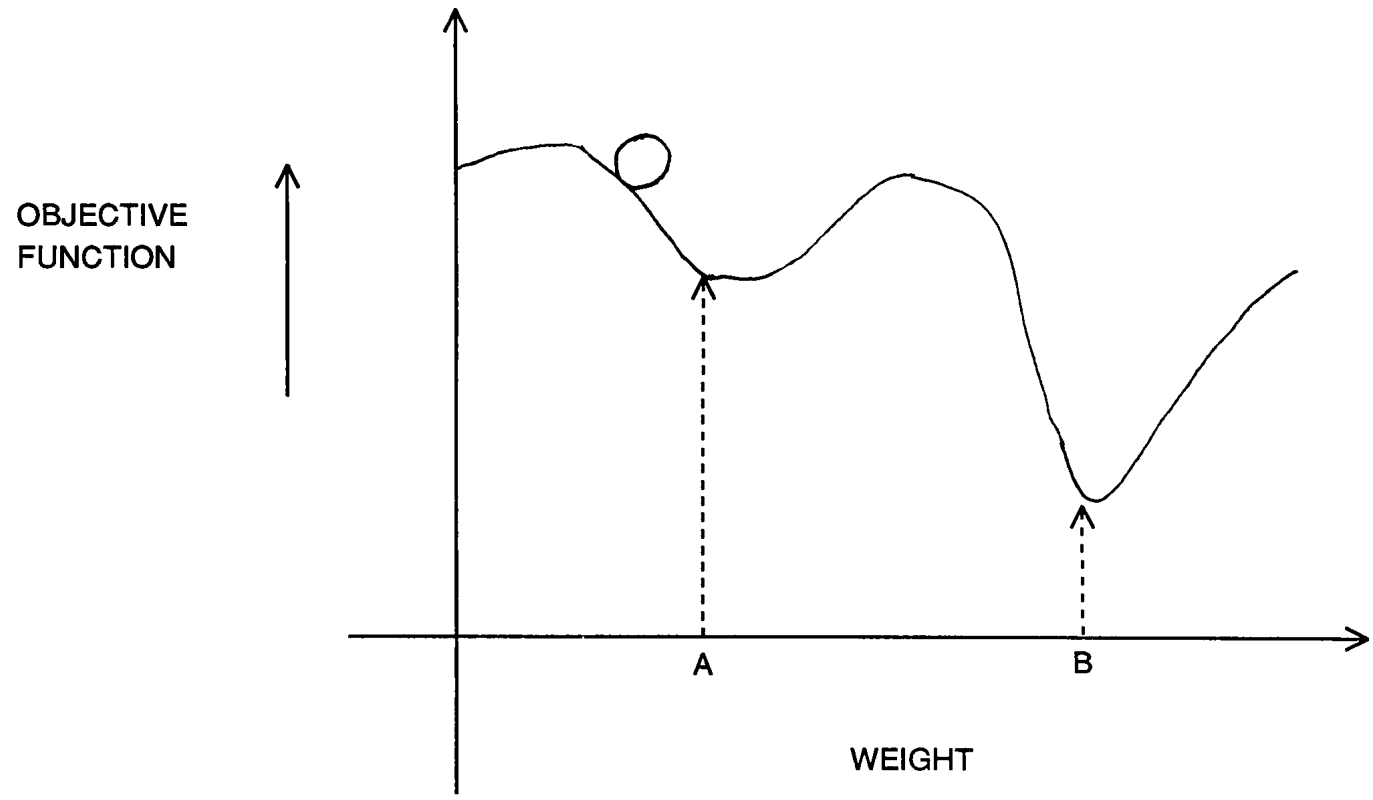
Cauchy training is very similar to Boltzman training except a Cauchy distribution replaces the Boltzman distribution. In a Cauchy distribution, step sizes (changes in weight values) are increased resulting in decreased training time.

The networks that have been discussed to this point are non-recurrent networks, that is, there are no feedback connections from the network's output to the network's inputs. By not utilizing feedback connections, non-recurrent neural networks are always "stable". With recurrent networks the output(s) are fedback to the input(s) and the output(s) are recalculated (fig. 18) [17]. The process is repeated until the output(s) become constant.

There are many different types of neural networks and network training algorithms, of which some basic types have been discussed. Each approach is suited to solve some particular class of problems, so not all are suited for use in the solution of the shortest obstacle free path.

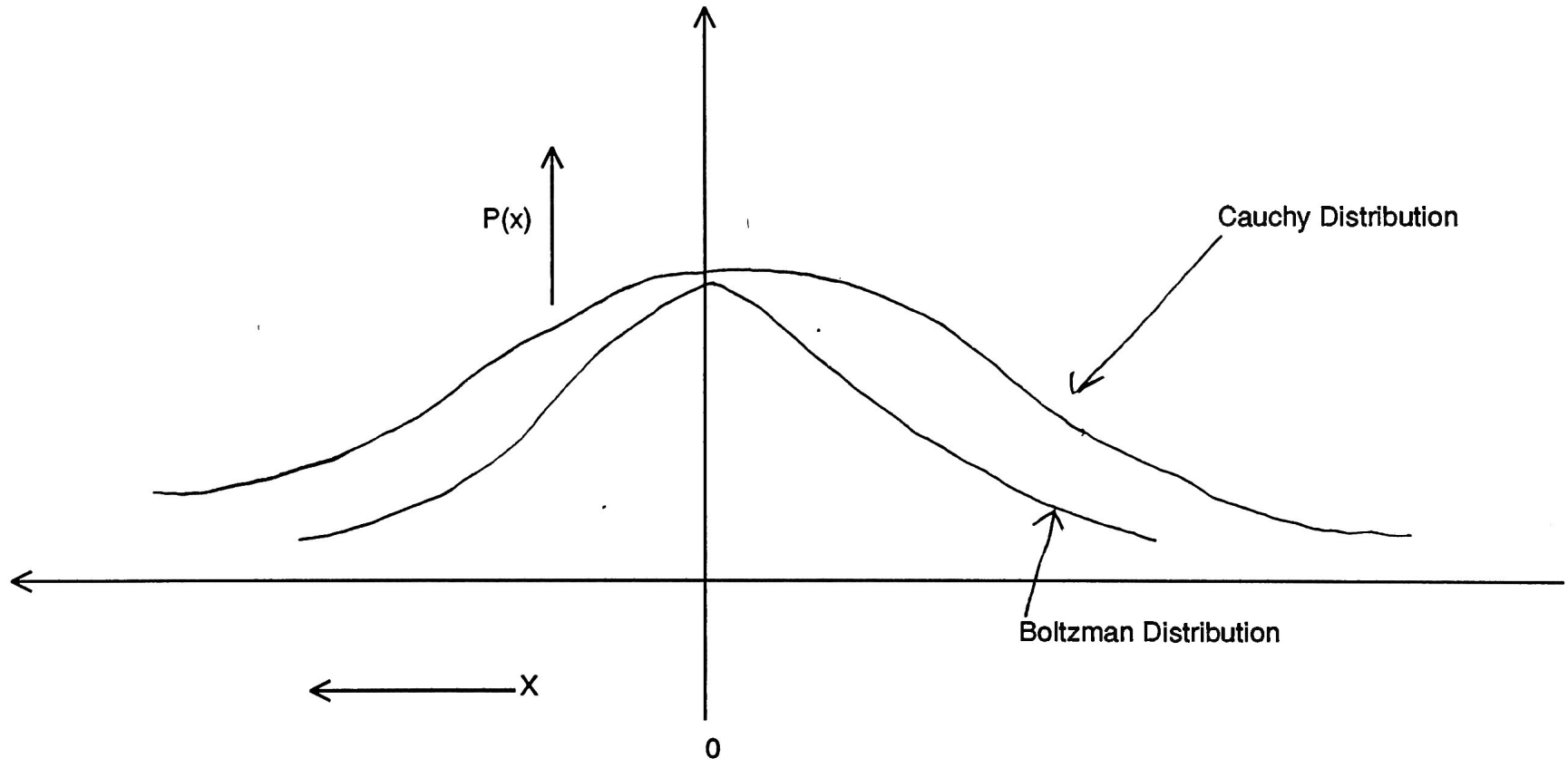
When trying to solve the shortest obstacle free path problem (S.H.O.P) we do not know what the desired output is

16;3
FIG. 16 - Local Minimum Problem [15]

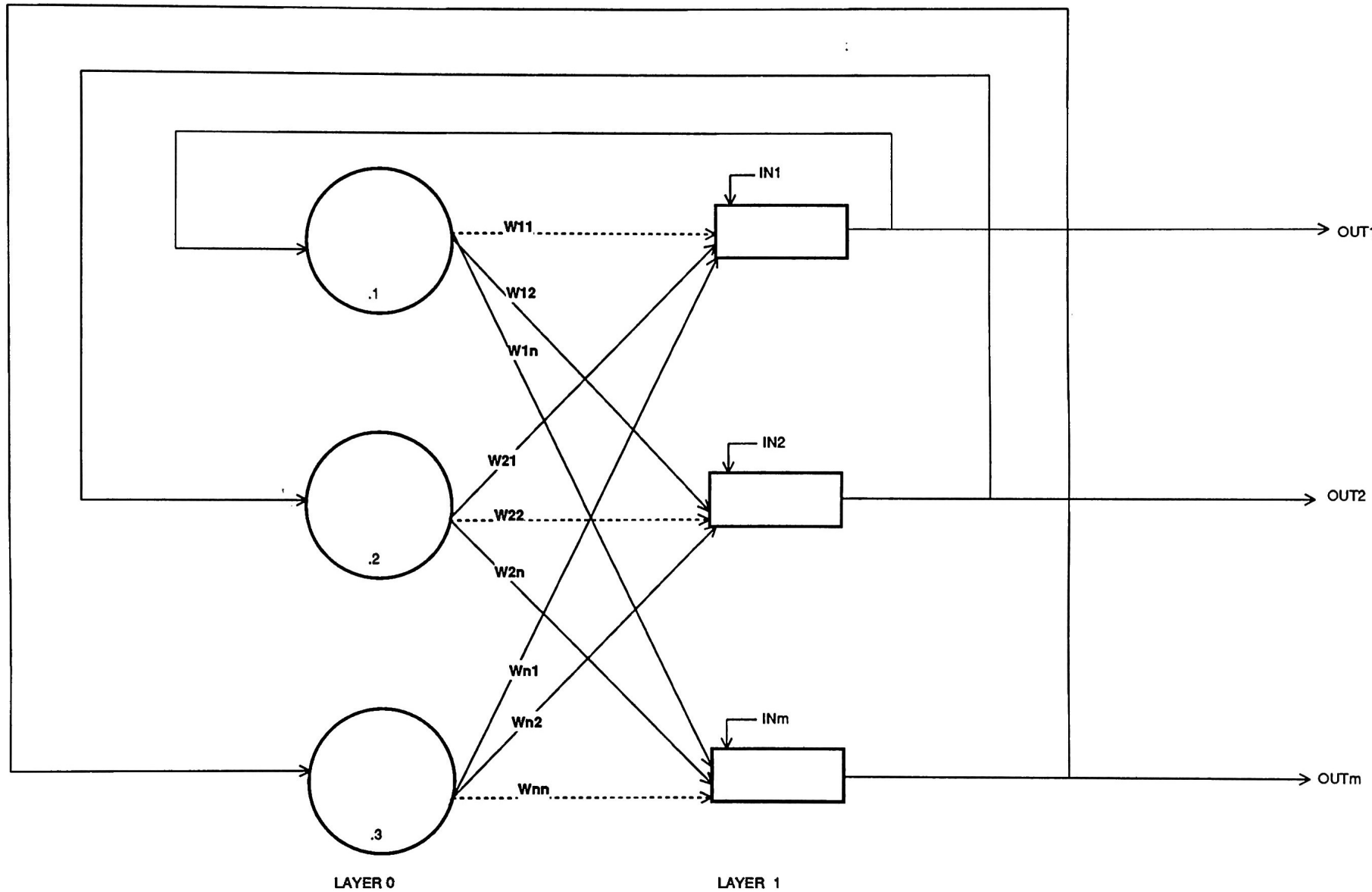


17;3

FIG. 17 - Cauchy versus Boltzman Distributions [16]



8;4
IG. 18 - Single Layer Recurrent Network [17]



** Dotted lines indicate weights of zero

as it is not a classification problem but a minimization problem. This then leads to the exclusion of "supervised" training algorithms as they require the expected output to be known. As for the actual structure of the network, a non-recurrent network is preferred as it is a more "stable".

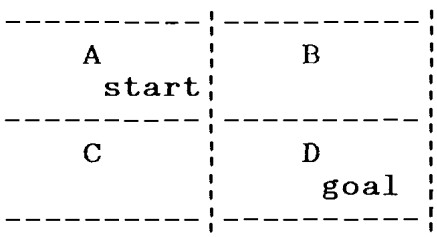
Although both a global planner and a local planner have been discussed, the global planner's function of finding the best path can be easily transferred to the network and will be the only path planner addressed here. We will assume the local planner is another program having inputs into the neural network that represents the global planner.

The path planning approach discussed earlier in which the robot's configuration space is broken into N-evenly sized cells can be implemented using nodes in the neural network to represent cells in the pseudo configuration space. The unsupervised training would be applied to the S.H.O.P application. Boltzman training is an example of an unsupervised training technique used when attempting to minimize an objective function's value. Since S.H.O.P is a minimization problem it will be the Boltzman training technique that will be explored first.

USING NEURAL NETWORKS FOR PATH PLANNING

A Boltzman neural network was created for the configuration space in FIG 19. This network (FIG 20) consists of an input layer with one processing element (PE)

FIG. 19



Robot Config. Space

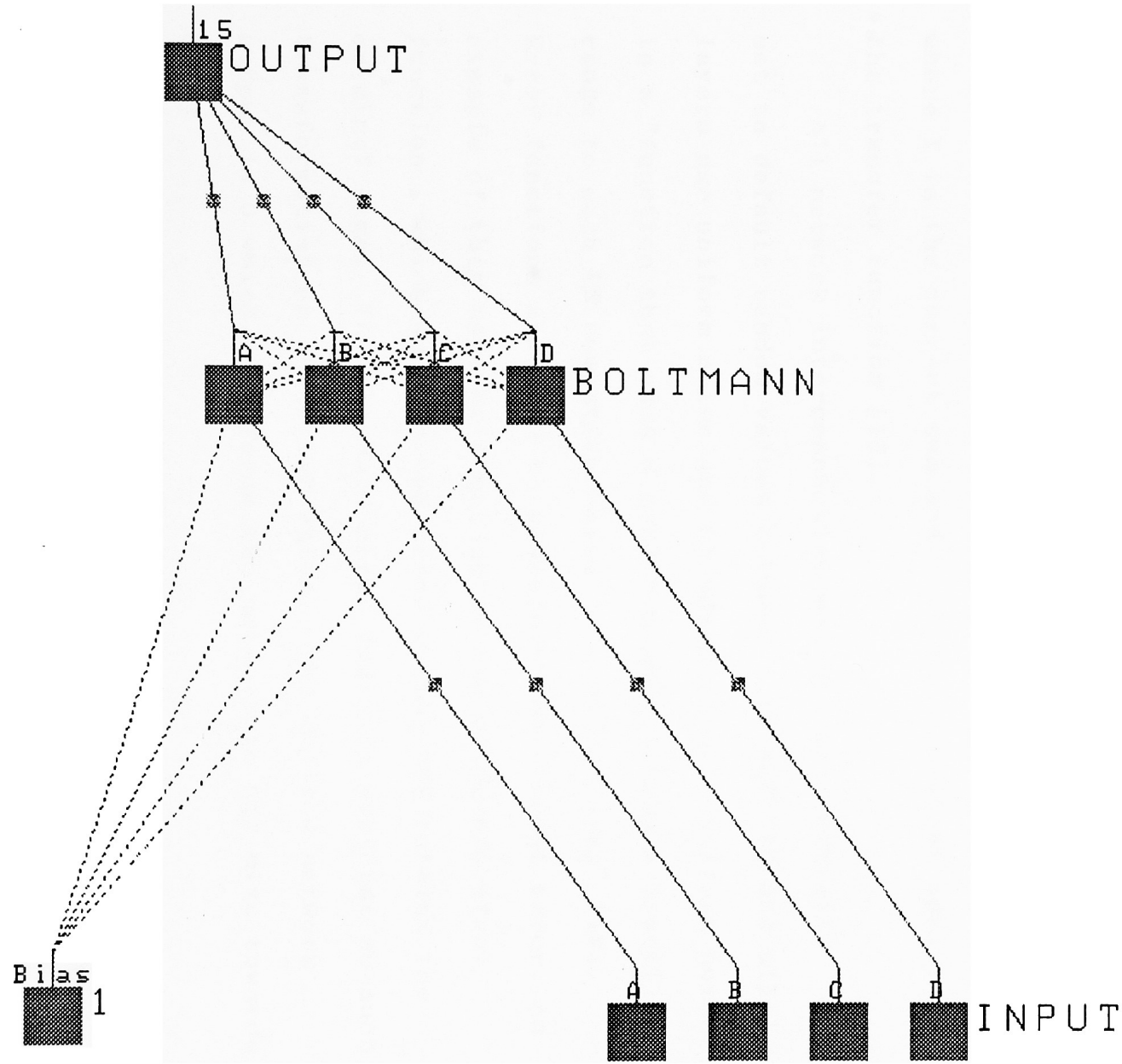
Cell	X	Y	Va	Vb
A	12	10%	0	2
B	22	20%	1	1
C	17	15%	1	1
D	7	5%	2	0

for each cell in the configuration space. There are 4 cells in the configuration space in FIG 19 so there are 4 PE's in this layer. Each input represents a configuration cell's value X where X's value is calculated by adding the distance to the start cell (Va) plus the distance to the goal cell (Vb) and the percent the cell is filled with an obstacle (Y) (X = Va + Vb + Y).

This algorithm was choosen as it simply represents the distance in the configuraton space and takes into account the existance of obstacles. Although in a very large configuration space, cells with no obstacles in them would have the same value as those that are closer to the start and goal positions but are 100% filled with an obstacle, the formula was simple to develop and use.

Next is the Boltzman layer with one PE (Processing Element) for each cell in the configuration space where each

Boltzman Network - FIG 20



OK to delete the network?(y/N): y
I 1605 Network <FIG20.nnd> successfully saved.

PE in the input layer is connected to every PE in the Boltzman layer. This layer uses Boltzman training on the sum of the inputs to this layer and PE's use the Boltzman transfer function.

The last layer is the output layer consisting of one PE where each PE in the boltzman layer is connected to this output PE. This layer uses no training algorithm on the sum of the inputs to this layer and PE's use the step function transfer function. The step transfer function is:

```
T = 1 , if I > 0
    0 , if I <= 0
```

where I is the current sum and T is the result of applying the transfer function [18].

All network interconnection weights are initially set to default random values between -.10 and .10 and all layers use uniform noise and standard error. Uniform noise is a "function that adds a random number within a specified range to each PE summation value in a given layer" [19]. Error functions are used to transform the current error. An example of this type of function is the quadratic error function , which squares the error value, but retains the original sign. The standard error function performs no such transformation on the error value. The desired network output is 0 which will force the network to minimize towards 0.

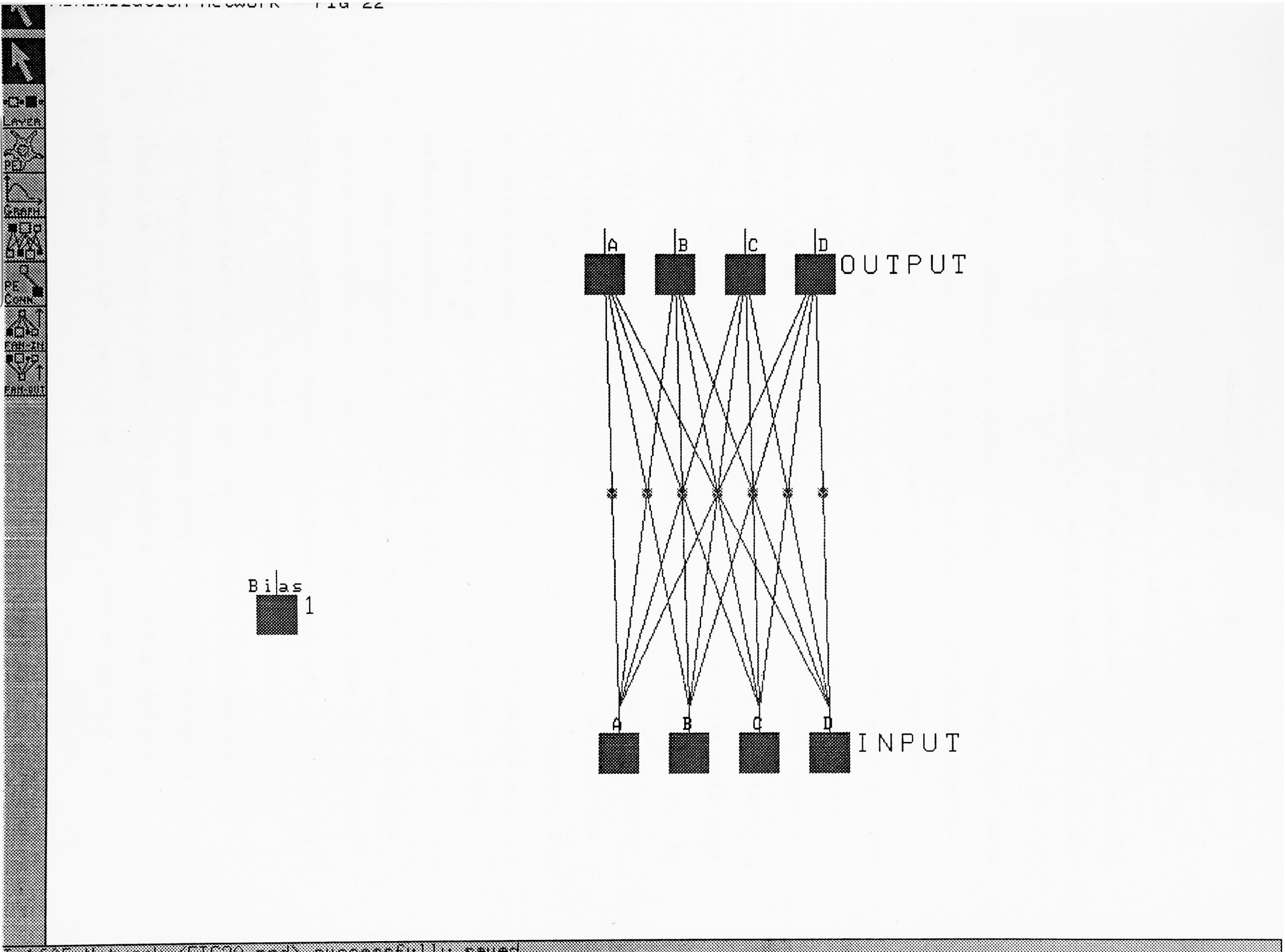
The training data shown in FIG 21 (same 4 cell configuration space as shown in FIG 19) was run thru the

network (using the same formula described previously for the value of input X) for several hundred epoches. The network FIG. 21

Cell	A	B	C	D
X value	12	22	17	17
	7	12	22	4
	52	32	22	14
	51	32	32	14
	18	6	3	11

interconnection weights changed as training progressed. A Neuralware tool that allows looking at a PE's value to determine that the network interconnection weights were in fact changing. The training was stopped at about 700 epoches and test data (A=12,B=22,C=17,D=7) was run thru the network. Determination of what the minimum path was from the network's output and the PE states was not possible (Appendix A1).

The paper entitled "Fast Computation of Optimal Paths in the Two- and Higher Dimension Maps" by Mohamad H. Hassoun and Ashvin J. Sanghvi [20] describes a method of solving the optimal path problem by using a series of minimization subnets for each processing element. These minimizations are done in parallel and network interconnection weights are held constant at 1. Based on this general principle a 2 layer minimization network was created (FIG 22) where each PE in the input layer is connected to every PE in the output layer, there is no hidden/Boltzman layers, and the output layer has one PE for each cell in the configuration space and determines the



I 1605 Network <FIG20.nnd> successfully saved.
I 1605 Network <FIG22.nnd> successfully saved.

minimum of it's inputs. The PE's in this layer use no transfer function (result of the minimizations is the output) and the network interconnection weights are held constant at 1.

The configuration space in FIG 19 was used and test data (A=12,B=22,C=17,D=7) was run thru the minimization network. Since all input PE's were connected to all output PE's, the solution presented via the network outputs, were all the same (all outputs were equal to 7 which is cell D's value). This is because the solution contained invalid North East , South East, North West and South West robot movements (only North, South, East and West valid), so the shortest path (but not a valid one) was from starting cell A to goal cell D. In addition, it was difficult to determine the network outputs from looking at the graphical representation of the network as all output cells were activated (solid box). Instead, the network's output file has to be printed. This is not a "true" neural network in that no training was done on the network interconnection weights (held constant at 1).

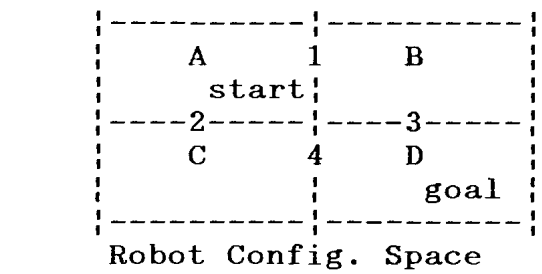
The two problems uncovered in the minimization network so far are that there is a difficulty in seeing the network results graphically (want nodes that are in the solution path to be the only ones active) and that the network uses no training algorithm (ex: Boltzman, Cauchy training etc.).

Because initial research showed that a Boltzman network should be used to solve the S.H.O.P. problem, another Boltzman network was tried that was similar to FIG. 20. This

was to address the problem of not developing a "true" neural network. The difference between this new Boltzman network and the first one presented in FIG. 20, is that the inputs to the network would be the "cell connection" values and not the values of the cells themselves (FIG. 23)

The "cell connection" value is obtained from the cell the robot would move into if it crossed that connection (moving from start to goal cells).

FIG. 23



Robot Config. Space

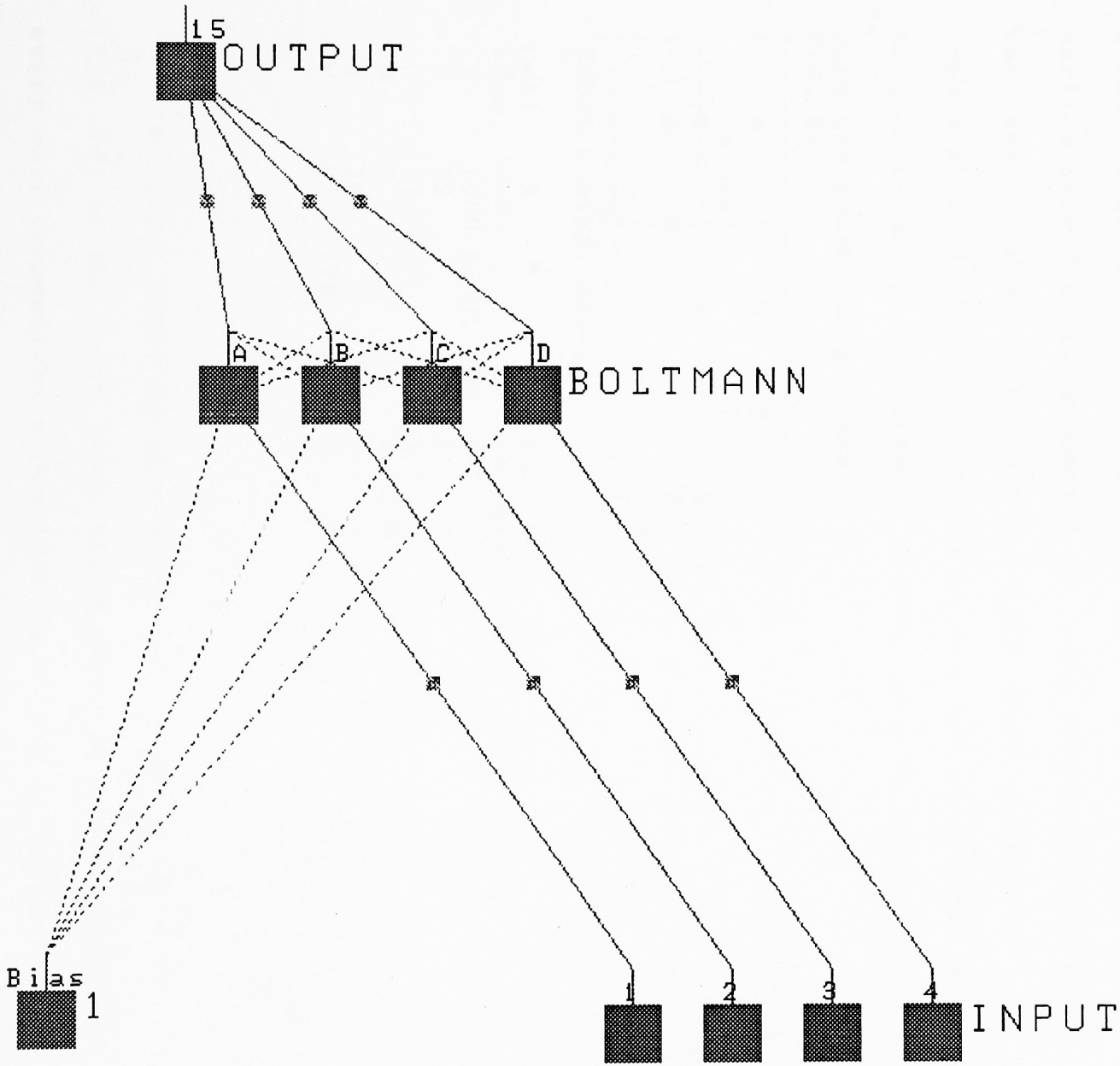
Cell	X	Y	Va	Vb	Cell Conn
A	12	10%	0	2	1=22 (B)
B	22	20%	1	1	2=17 (C)
C	17	15%	1	1	3=7 (D)
D	7	5%	2	0	4=7 (D)

This second Boltzman network (FIG. 24) consists of an input layer with one PE (Processing Element) for each "cell connection" in the configuration space. Where each input represents a configuration cell's connection value X.

The Boltzman and output layers are the same as those found in the first Boltzman network (FIG. 20) with the exception that the only connections made between the PE's of the input and Boltzman layers are the valid robot movement connections (IE: North,South,East,West).

This new Boltzman network was trained using the data in FIG. 25 for about 700 epoches, then test data (1=22,2=17,3=7,4=7) was run thru the network, but the results were the same as for the first Boltzman network (Appendix A2).

Boltzman Network - FIG 24

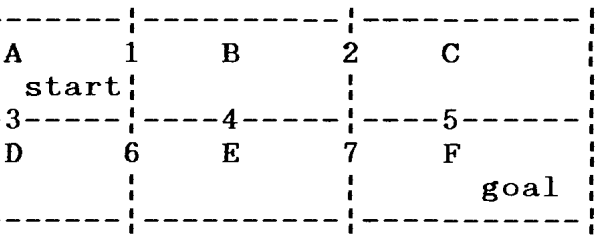


OK to delete the network?(y/N): y
Boltzman Network (FIG24.prd) successfully saved.

In retrospective, the reason why the results were no different than that from the first Boltzman network was because the representation of the objective function was incorrect. The only thing changed in the second Boltzman network was the inputs but not the representation of the function. Consequently, the same problems were found.

No success had been found using the Boltzman networks (true neural networks). The minimization network discussed earlier proved somewhat useful so a new minimization network was created. First, the configuration space and network inputs shown in FIG. 25, would be used and an attempt will be made to design the network so the cells in the solution path are graphically visible.

FIG. 25

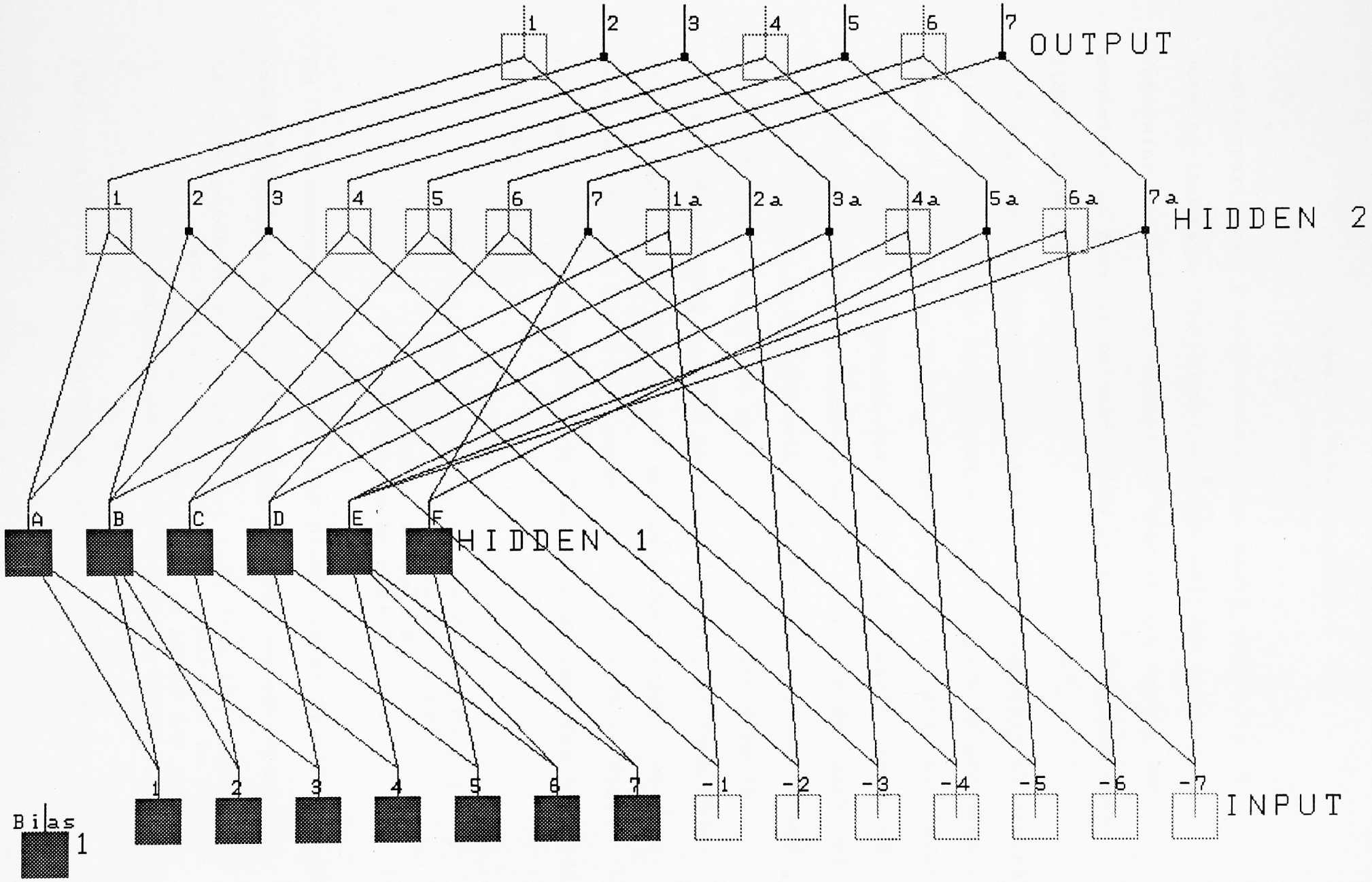


Robot Config. Space

Cell	X	Y	Va	Vb	Cell Conn
A	1000 (12)	10%	0	2	1=22 (B)
B	22	20%	1	1	2=17 (C)
C	17	15%	1	1	3=7 (D)
D	7	5%	2	0	4=30 (E)
E	30	27%	2	1	5=0 (25) (F)
F	0 (25)	22%	3	0	6=30 (E)
					7=0 (25) (F)

The first layer in this second minimization network (FIG. 26) is the input layer which consists of two inputs for every cell connection, a positive input and it's negative

Minimization Network - FIG 26



I 1605 Network <FIG22.nnd> successfully saved.
I 1605 Network <FIG26.nnd> successfully saved.

counterpart (ex: 7 cell connections would result in 14 network inputs). The negative input will be used to determine which "cell connection" was chosen during the minimization process (minimization process takes place in Hidden Layer 1).

Next is the first hidden layer, which consists of one PE for each cell in the configuration space. Each PE performs a minimization on it's inputs. Again (as in FIG 22) this layer uses no training algorithm and PEs use no transfer function.

The second hidden layer consists of one PE for each cell connection (ex: in FIG 25 cell A has two connections, 1 and 3). This layer receive's as it's input outputs from the first hidden layer, and the negative inputs (-X) from the input layer. Each PE in this second hidden layer, can only have two inputs, the first connection is to a PE in first hidden layer (corresponds to a "cell connection" for the cell in hidden layer 1).

ex: Hidden Layer 1 Cell A = Config Cell A
Cell A has two "cell connections"
1 = connection to cell B
2 = connection to cell C

So "cell connections" 1 and 2 (in hidden layer 2) will get an input from Cell A in hidden layer 1 . The second connection to corresponds to a negative input (-X) from the input layer.

ex: "cell connection" 1 = connection to cell B
Will receive as input -X value for cell connection 1 from the input layer.

This restriction is required so that a configuration cell

choosen as the minimum of the inputs can be identified by adding in the corresponding negative number which results in a sum of zero. For example if configuration cell A was choosen as the minimum, then it's value X would be propogated forward thru the network. To determine that cell A was choosen, the output X is summed with the input -X from where a sum of zero indicates the cell was choosen. If more than two inputs were allowed, this approach would not work.

The PE's in this layer perform a summation on their inputs (if the sum is zero then the cell connection is in the solution path). Again, no training algorithm is used and the transfer function is signum 0. The signum 0 transfer function is:

$$T = \begin{cases} 1, & \text{if } I > 0 \\ 0, & \text{if } I = 0 \\ -1, & \text{if } I < 0 \end{cases}$$

where I is the current sum and T is the result of applying the transfer function.

The last layer is the output layer which contains one PE for each cell connection in the configuration space. The inputs to this layer are the corresponding PE cell connection outputs from hidden layer 2.

ex: "cell connection" 1 = connection to cell A
"cell connection" 1a = connection to cell B
-Both these outputs from hidden layer 2 are fed into the output layer PE cell for "cell connection" 1.

Each PE in this layer performs a max on it's inputs and uses no transfer function. If the max is equal to zero the "cell

connection is in the solution path (0 is graphically represented as a dot, 1 is a solid box ,and -1 is a clear box). The output layer uses no training algorithm and is needed because if any PE for a particular cell connection is a 0, then the cell is in the solution path. One example is cell connections 5 and 5a. In hidden layer 2, PE 5's value is -1 and PE 5a's value is 0. Since one of the two PEs output value is zero, then the connection is in the solution path. If the maximum of these two inputs to the output layer is taken it is 0.

All interconnection weights are held constant at 1 and the network output is a set of -1 and 0 values, where 0 indicates which cell connection in the configuration space would be in the solution path (the solution path, then becomes graphically visible). Since the start cell is assumed to be in the solution path it's value need not be identified as a zero.

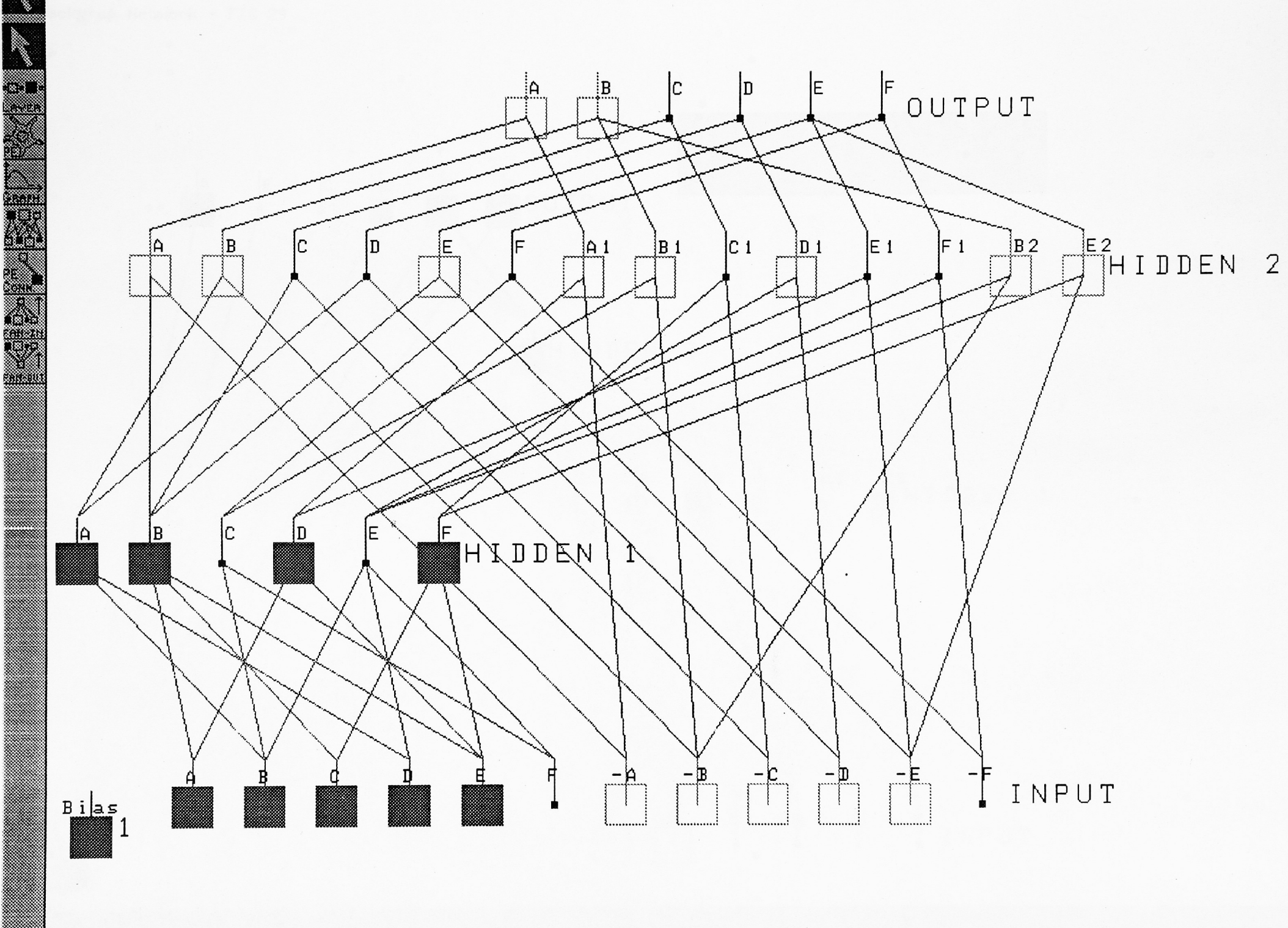
The problem this solution presents is that some of the cells identified result in "dead end" paths. In some cases no path that is not a "dead end" is found. For example, when a test was run using the data for the 6 cell configuration space (FIG. 25) there were 4 cell connections whose corresponding PE output values were zero (so therefore have been identified as cell connections in the solution path). The cell connections identified were: 1,2,3,5 and 7 (PE 1's value is not equal to 0 but since it is the start cell it is assumed to be in the solution path). If the robot starts at

cell A and cross over connection 3 the robot can go no further as the next connection to cross is 6 and it was not one of the cell connections identified above (this is a "dead end" path). This problem is created because in hidden layer 1 the network calculates that the minimum between two different cells as the same value (ex: from cell A go across connection 3, from cell D go across connection 3, so will never get to connection 6).

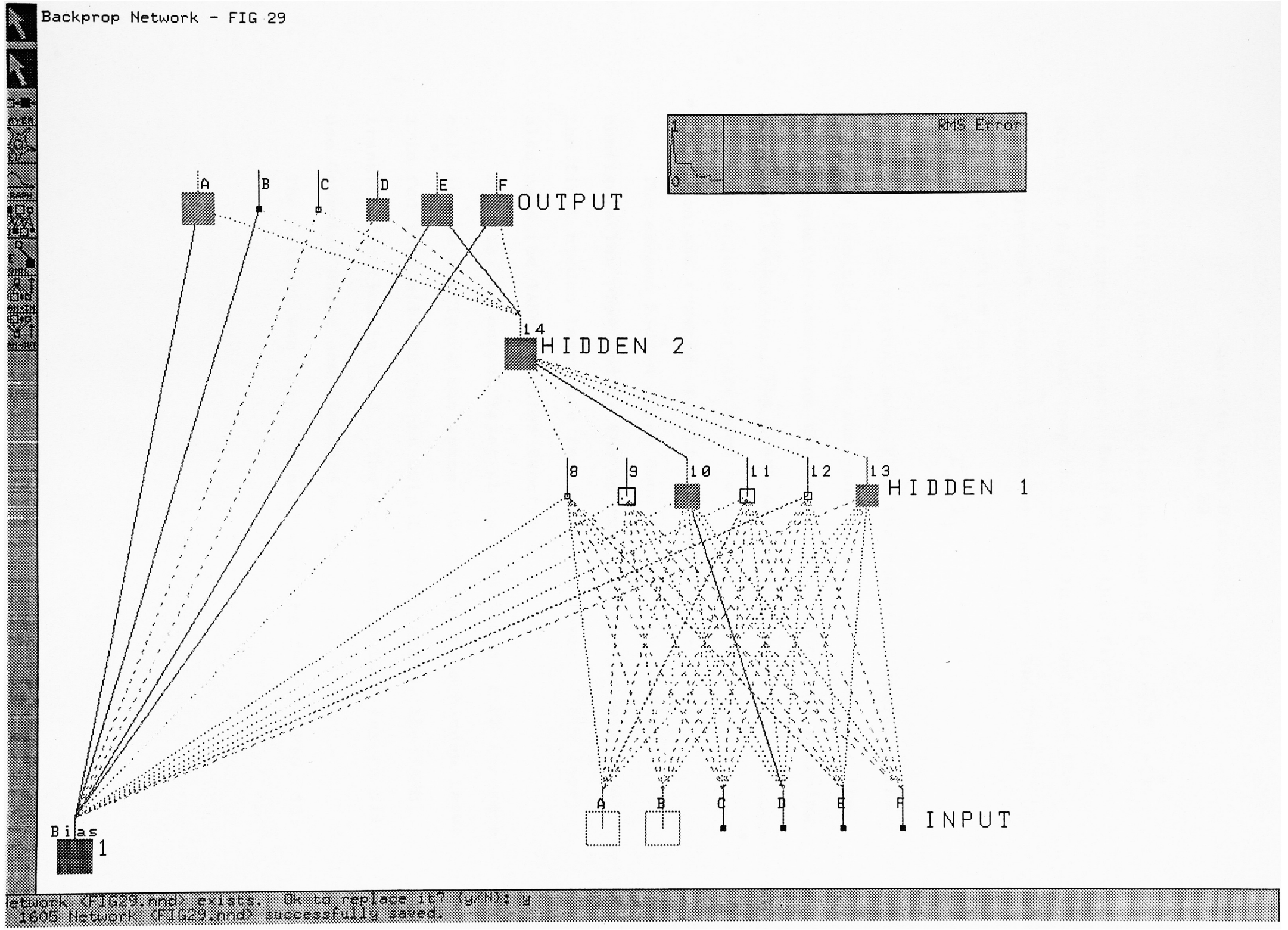
To help eliminate situations were no valid path is found (all paths found are "dead end" paths), a network was created that uses the original configuration space representation where the cell values , not the cell connections, are used as inputs (FIG 28). This eliminated the problem in which no valid path was found, but some invalid paths were still chosen.

Since the output from the minimization network (FIG 28) is in 0's and -1's, it can be fed into a backpropagation network that has been trained to classify "valid paths". The restriction would be that the start and goal cells would have to remain constant, the only variables that could change is the percent each cell is filled with an obstacle (without having to re-train the backpropagation network).

A backpropagation network was created (FIG. 29) in which there is a input layer with 6 PE's one for each cell in the configuration space (FIG. 25). The input to this network is the output from the minimization network created in FIG 28.



OK to delete the network?(y/N): y
I 1605 Network <FIG28.nnd> successfully saved.


Backprop Network - FIG 29
RMS Error
1
0
A
B
C
D
E
F
OUTPUT
14
HIDDEN 2
8
9
10
11
12
13
HIDDEN 1
A
B
C
D
E
F
INPUT
Bias
1
etwork <FIG29.nnd> exists. Ok to replace it? (y/N): y
1605 Network <FIG29.nnd> successfully saved.

The first hidden layer also has one PE for each cell in the configuration space. Each PE in this first hidden layer is fed each input from the input layer and uses the TANH (hyperbolic tangent) transfer function. The TANH transfer function is:

$$I' = I * GAIN$$
$$T = (e^{I'} - e^{-I'}) / (e^{I'} + e^{-I'})$$

Where I is the current sum, T is the result of applying the transfer function to the sum, and Gain is the value for the gain parameter taken from the recall section of the learning and recall schedule. The learning and recall schedule contains values that are used to control the summation and transfer function's learning rate.

The second hidden layer consists of 1 PE (default configuration provided by the system) and all outputs from the first hidden layer are fed into this layer. This layer also uses the TANH transfer function.

Next is the output layer which consists of 1 PE for each cell in the configuration space. The output from hidden layer 2 is fed into all PEs in the output layer, again, the TANH transfer function is used. The hidden and output layers all use Gaussian noise and standard error.

The network was then trained using the data in FIG. 30.

FIG. 30 - Training Data

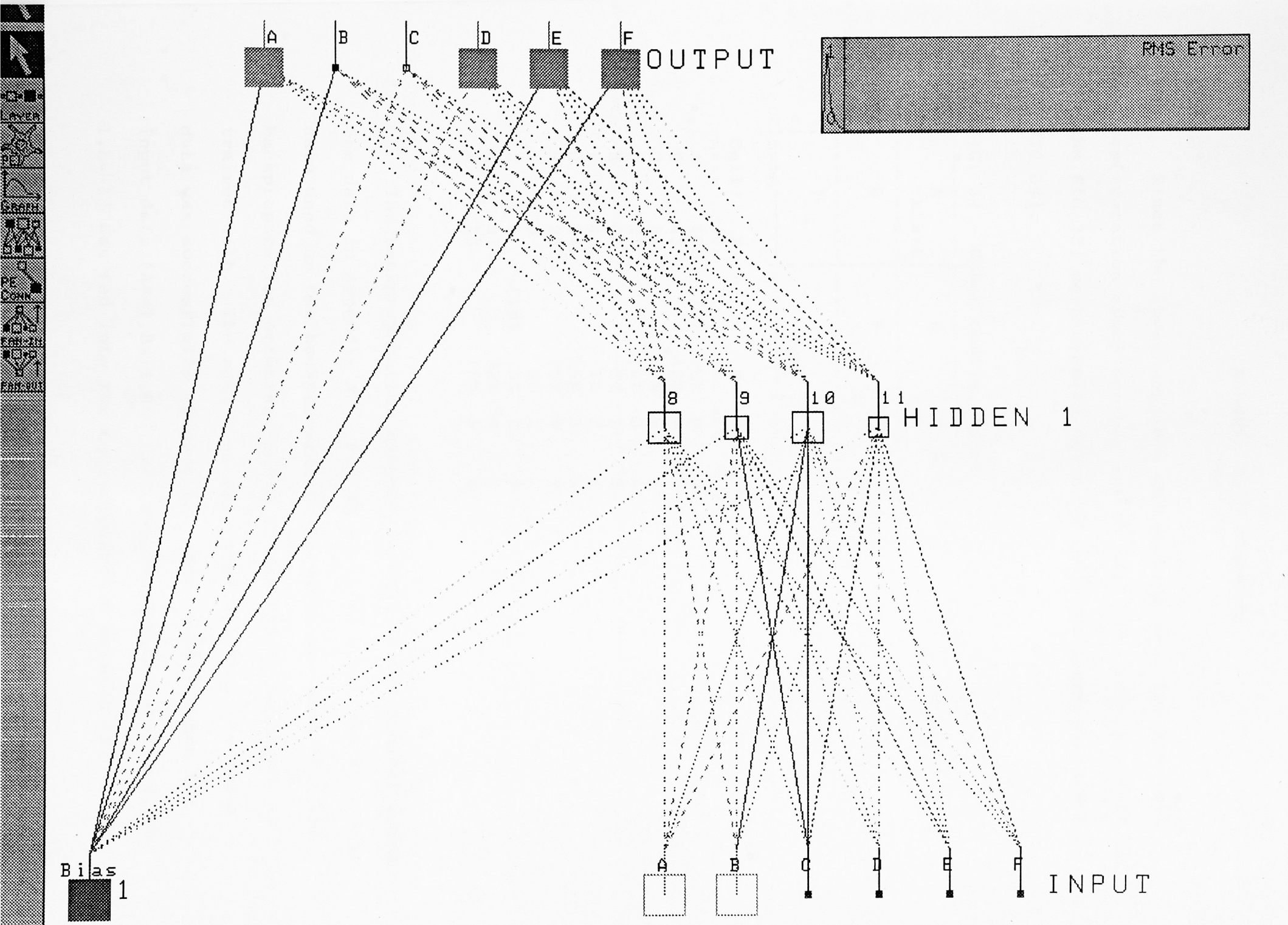
	Input from Minimization Network						Expected Output					
Cell	A	B	C	D	E	F	A	B	C	D	E	F
	-1	-1	0	0	0	0	1	0	0	1	1	1
	-1	0	-1	-1	0	0	1	1	0	0	1	1
	-1	0	0	-1	0	0	1	1	1	0	0	1

The training data inputs were obtained by running various data values thru the minimization network in FIG 28, then recording the network's output. This output was then identified as belonging to a class of outputs (used same start and goal cells).

This network's error leveled off at about 0.25 around 2100-2300 training epoches. Training is complete when the network's RMS error is zero or close to zero this can be seen in the RMS graph in FIG. 29.

To reduce the number of training epoches and to get the RMS error as close to zero as possible, the backpropagation network in FIG. 29 was modified to reduce the number of PEs in the hidden layer to 4 and to remove the second hidden layer (FIG 31). The result is a network whose RMS error is less than 0.25 (desired error is 0.0), accomplished in 500-700 training epoches (FIG 31).

This new backpropagation network was then tested using data from the training set (A=-1,B=-1,C=0,D=0,E=0,F=0) and the correct path was chosen by the network.



Network <FIG31.nnd> exists. Ok to replace it? (y/N): y
I 1605 Network <FIG31.nnd> successfully saved.

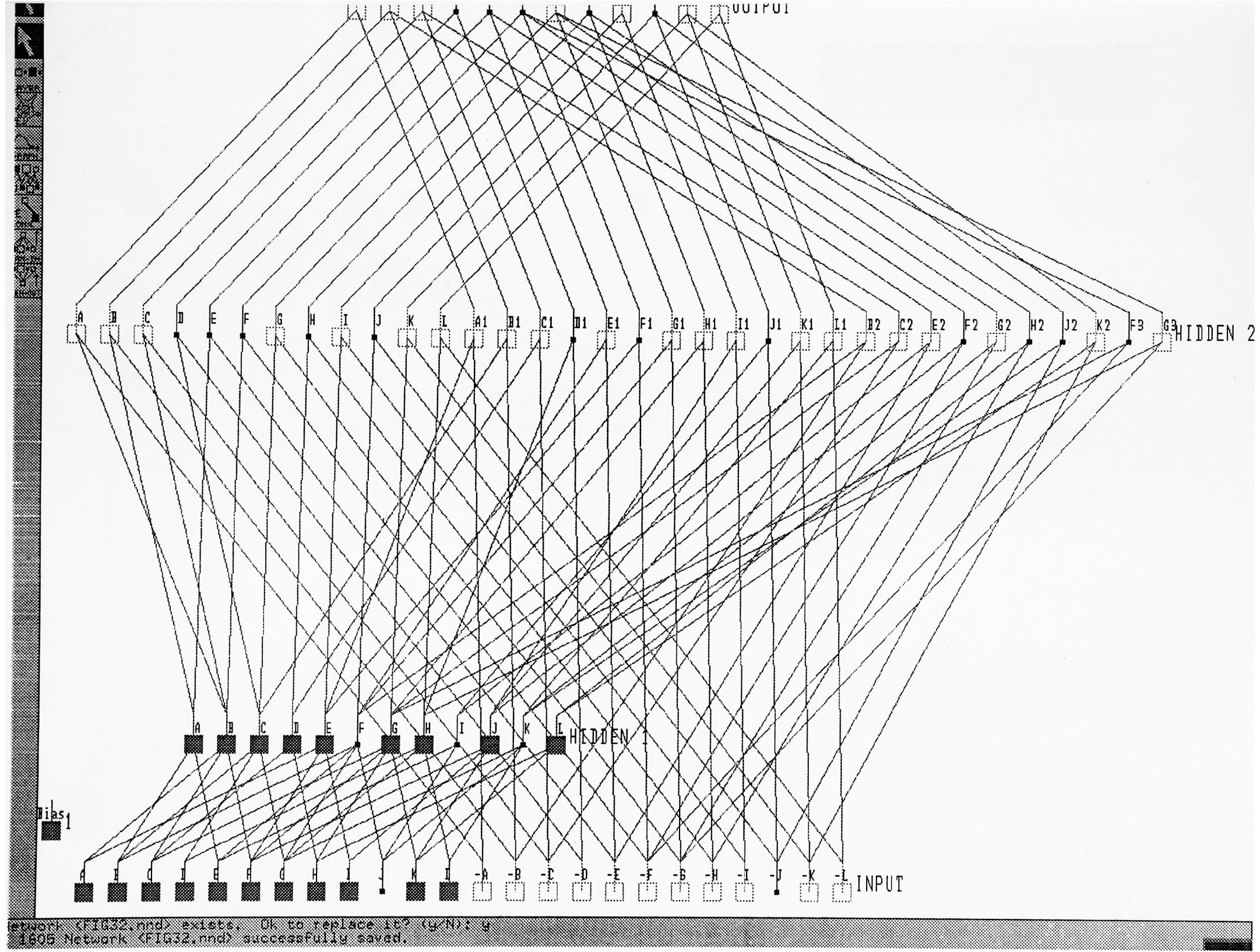
Since the above approach was working well for a 6 cell configuration space another set of duplicate networks (FIG 32 and FIG 33) were created for a 12 cell configuration space FIG 34).

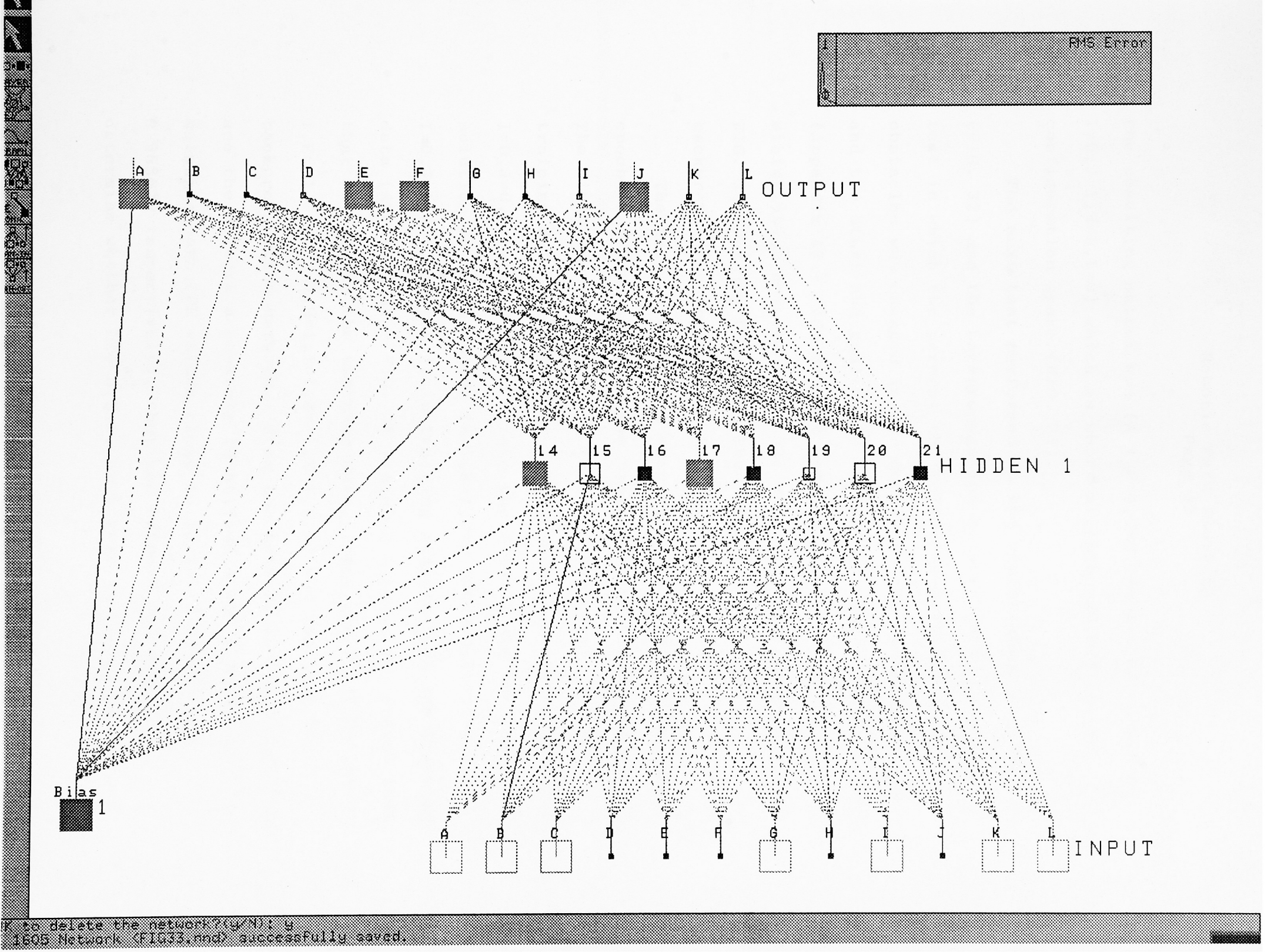
FIG. 34 - Robot Config. Space

A start	B	C	D
E	F	G	H
I	J goal	K	L

Cell	X		Y	Va	Vb
A	1000	(15)	12%	0	3
B	45		42%	1	2
C	34		29%	2	3
D	21		14%	3	4
E	5		2%	1	2
F	4		1%	2	1
G	34		29%	3	2
H	23		16%	4	3
I	9		6%	2	1
J	0	(12)	9%	3	0
K	37		32%	4	1
L	23		16%	5	2

The backpropagation network in FIG. 33 was trained using the data in Appendix A3 - FIG 33 Training Data. The only test data used on the backpropagation networks was data that the backpropagation networks in FIG 31 and FIG 33 had been trained with. The result of this test was that the input data was successfully classified. For example, when the input data (A=-1,B=-1,C=0,D=-1,E=0,F=0,G=0,H=0,I=-1,J=0,K=-1,L=-1) was fed into the backpropagation network (FIG. 32)


OUTPUT
A B C D E F G H I J K L A1 B1 C1 D1 E1 F1 G1 H1 I1 J1 K1 L1 B2 C2 E2 F2 G2 H2 J2 K2 F3 G3
HIDDEN 2
A B C D E F G H I J K L
HIDDEN 1
Bias 1
A B C D E F G H I J K L -A -B -C -D -E -F -G -H -I -J -K -L
INPUT
etwork (FIG32.nnd) exists. Ok to replace it? (y/N): y
1605 Network (FIG32.nnd) successfully saved.

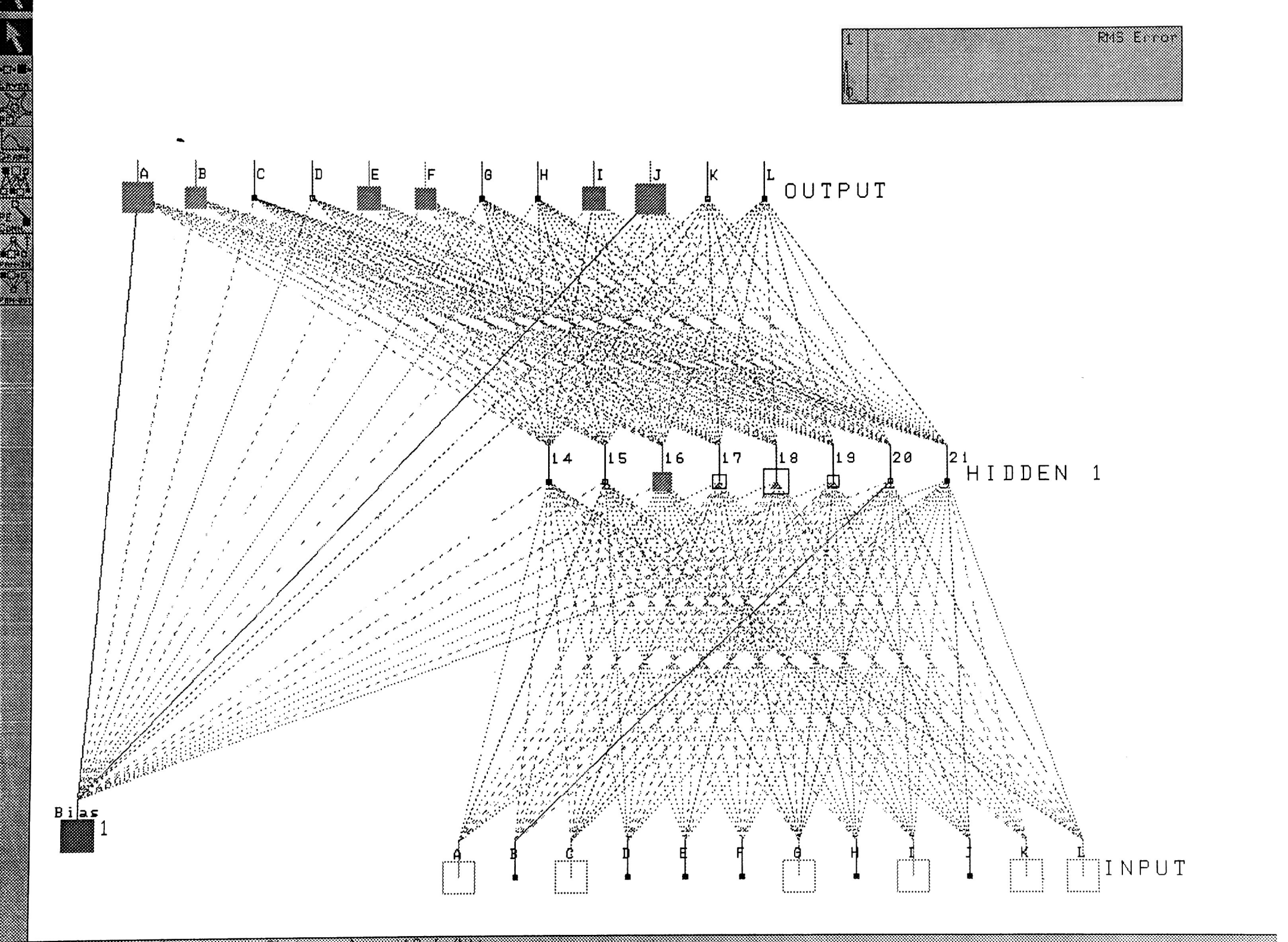


K to delete the network? (y,N): y
1605 Network <FIG33.nnd> successfully saved.

the resulting output was (A-1,B=0,C=0,D=0,E=1,F=1,G=0,H=0, I=0,J=1,K=0,L=0) which is shortest path A,E,F,J in the configuration space (FIG. 34).

The next test performed on the minimization network (FIG. 32) and the backpropagation network (FIG. 33) was a a test in which the percent the cell's were filled with obstacles was changed in the configuration space (FIG. 34) and the start and goal cells remained constant. This new data (Appendix A3 - FIG. 32 Test Data) was fed into the minimization network (FIG. 32) and the output (Appendix A4 - FIG. 35 Training Data) was then used to train the backpropagation network in FIG 33.

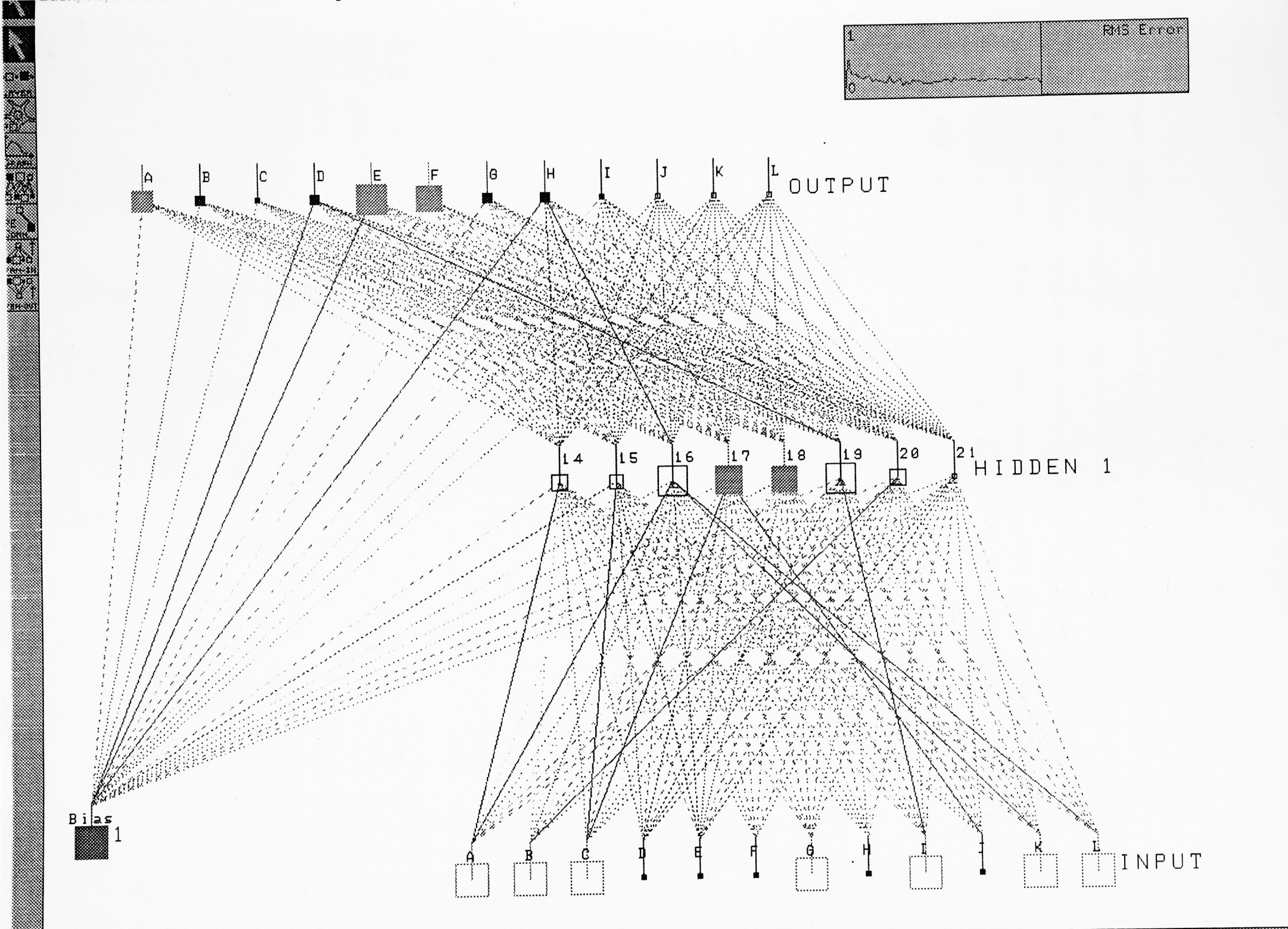
The network error was very close to 0 after about 1000 epoches and two tests on the trained network were performed. The first was to test input that had been part of the training data, so input (A=-1,B=0,C=0,D=-1,E=-1,F=-1,G=0,H=0, I=0,J=0,K=0,L=-1) was fed into the network and the correct solution path was chosen (A=1,B=1,C=0,D=0,E=0,F=1,G=0,H=0, I=0,J=1,K=0,L=0) as it is training example 5 in the training data (APPENDIX A4 - FIG. 35 Training Data). The second test that was not in the training data, input (A=-1,B=0,C=0,D=-1,E=-1.F=-1,G=1,H=1,I=1,J=-1,K=1,L=1) was fed into the backpropagation network. From the network in FIG.35, you can see the network accurately identified A and J as being in the solution path (PE values close to 1, Ie: 0.970621 and 0.963629 respectively), but the network was unable to determine whether PEs B,E,F, or I were in the solution path



etwork <FIG35.nnd> exists. Ok to replace it? (y/n): y

as all these PEs had values around 0.7 . This problem could be corrected by creating a larger training set for the backpropagation network, investigation and testong of this theory is not included in this paper.

The next experiment was to training the backpropagation network in FIG. 33 with data in which the percent the cell was filled with an obstacle remained constant and the start and goal cells changed. The test data in APPENDIX A4 -FIG. 32 (changing position of start and goal cells) was fed into the minimization network in FIG. 32. The result was several outputs that were the same but should represent different paths. The result of this was training data in which some of the inputs were the same with different desired outputs (see APPENDIX A5 - FIG. 37 Training Data examples 4 and 5). The training data was then fed into the backpropagation network in FIG. 33 and a test was run using one of the training examples whose input was duplicate in the training set input #4 (A=-1,B=-1,C=-1,D=0,E=0,F=0,G=-1,H=0,I=-1,J=0,K=-1,L=-1), the result can be see in FIG. 37. Because multiple training data set entries were the same with different desired outputs, the backpropagation network was unable to determine what the solution path should be. In the example discused above, the correct path should be A,D,E,F,G,H, but the network was only able to correctly identify PE E whose value is 0.986 (see FIG. 37), PEs A,B,D,E,F,G,H had values between 0.3 and 0.8. A solution to this problem would be to have separate training sets for every possible location of start



etwork (FIG37.nod) exists. OK to replace it? (y/n): y

and goal cells. Once the start and goal positions were known, the correct trained backpropagation network can be chosen that is trained with different percent cell filled values but constant start and goal cell positions. The network selected would be similar to that discussed above in FIG. 35.

CONCLUSION

In conclusion, the best solution for the S.H.O.P problem using networks is a combination of two types of networks, a minimization network whose output feeds into a backpropagation network. A series of trained backpropagation networks are required to handle situations in which the start and goal cell positions change. Another possible way of handling changes in the start and goal cell position would be to use the backpropagation network trained for different start and goal positions (FIG. 37) and send the best guess path to the robot. The robot would then try to execute this path and if it is unable to continue (encounters a dead end) the robot would communicate back to the planning program (consists of two neural networks previously discussed) that the path was not correct and why (ex: could not move from cell C to cell H). The planning program could then automatically change the percent cell C is filled to decrease the chances of this cell being chosen (use cell filled percentage as a probability that the cell is in the solution path).

Future work on the S.H.O.P problem using neural networks would be to include multiple trained networks for different cell start and goal positions (global planner), use of larger training sets when training the backpropagation networks and the integration of this global planner with a local planner that actually controls the robot.

APPENDIX A

Below is an index to the input and output files for the networks created and discussed in this paper.

PAGE	DESCRIPTION
A1	Figure 20 (Boltzman Network) Output Data
A2	Figure 24 (Boltzman Network) Output Data
A3	Figure 32 (Minimization Network) Test Data * 12 cell configuration space * No training data as no training is done * Data for changing [ercent cells filled with obstacles
A3	Figure 33 (Backprop. Network) Training Data * 12 cell configuration space
A4	Figure 35 (Backprop. Network) Training Data * 12 cell configuration space * Data for changing percent cells filled with obstacles
A4	Figure 32 (Minimization Network) Test Data * 12 cell configuration space * No training data as no training is done * Data for changing position of start and goal cells
A5	Figure 37 (Backprop. Network) Test Data * 12 cell configuration space * Data for changing position of start and goal cells

```
Title: Boltzman Network - FIG 20 Output Data
       Display Mode: Network                          Type: Hetero-Associative
      Display Style: default
   Control Strategy: boltzio                  L/R Schedule: default
          0 Learn                27 Recall                2 Layer
         10 Aux 1                10 Aux 2                20 Aux 3
L/R Schedule: default
   Recall Step                1          0          0          0          0
     Input Clamp         1.0000     0.0000     0.0000     0.0000     0.0000
     Firing Density    100.0000     0.0000     0.0000     0.0000     0.0000
     Temperature         0.0000     0.0000     0.0000     0.0000     0.0000
     Gain                1.0000     0.0000     0.0000     0.0000     0.0000
     Gain                1.0000     0.0000     0.0000     0.0000     0.0000
   Learn Step             99999          0          0          0          0
     Coefficient 1       1.0000     0.0000     0.0000     0.0000     0.0000
     Coefficient 2       0.0000     0.0000     0.0000     0.0000     0.0000
     Coefficient 3       0.0000     0.0000     0.0000     0.0000     0.0000
     Temperature         0.0000     0.0000     0.0000     0.0000     0.0000
IO Parameters
         Learn Data: File Rand. (FIGAIN) Binary Load
        Recall Data: File Seq. (FIGAIN)
        Result File: Desired Output, Output
     UserIO Program: userio
         I/P Ranges:    -1.0000,     1.0000
         O/P Ranges:     0.0000,     1.0000
      I/P Start Col:          1              MinMax Table: default
      O/P Start Col:          5          Number of Entries: 0
MinMax Table <default>:
Layer: 1
        PEs: 1       Wgt Fields: 2                  Sum: Sum
    Spacing: 5        F' offset: 0.00            Transfer: Linear
      Shape: Square                               Output: Direct
      Scale: 1.00     Low Limit: -9999.00      Error Func: standard
     Offset: 0.00    High Limit: 9999.00           Learn: --None--
   Init Low: -0.100   Init High: 0.100       L/R Schedule: (Network)
   Winner 1: None                             Winner 2: None
   PE: Bias
       1.000 Err Factor      0.000 Desired
       0.000 Sum             1.000 Transfer         1.000 Output
           0 Weights         0.000 Error            0.000 Current Error
Layer: INPUT
        PEs: 4       Wgt Fields: 1                  Sum: Sum
    Spacing: 5        F' offset: 0.00            Transfer: StepFunction
      Shape: Square                               Output: Direct
      Scale: 1.00     Low Limit: -9999.00      Error Func: standard
     Offset: 0.00    High Limit: 9999.00           Learn: --None--
   Init Low: -0.100   Init High: 0.100       L/R Schedule: (Network)
   Winner 1: None                             Winner 2: None
   PE: A
       1.000 Err Factor     12.000 Desired
      12.000 Sum             1.000 Transfer         1.000 Output
           0 Weights        12.000 Error            0.000 Current Error
   PE: B
       1.000 Err Factor     22.000 Desired
      22.000 Sum             1.000 Transfer         1.000 Output
           0 Weights        22.000 Error            0.000 Current Error
   PE: C
       1.000 Err Factor     17.000 Desired
      17.000 Sum             1.000 Transfer         1.000 Output
           0 Weights        17.000 Error            0.000 Current Error
   PE: D
       1.000 Err Factor      7.000 Desired
       7.000 Sum             1.000 Transfer         1.000 Output
           0 Weights         7.000 Error            0.000 Current Error
```

```
Layer: BOLTMANN
        PEs: 4        Wgt Fields: 2                  Sum: S
    Spacing: 5         F' offset: 0.00           Transfer: Boltzmann
      Shape: Square                              Output: Direct
      Scale: 1.00     Low Limit: -9999.00     Error Func: standard
     Offset: 0.00    High Limit: 9999.00           Learn: Boltzmann
   Init Low: 0.000    Init High: 0.000      L/R Schedule: boltzman
   Winner 1: None                             Winner 2: None
L/R Schedule: boltzman
   Recall Step                6          11          16          21          26
     Input Clamp         0.0000      0.0000      0.0000      0.0000      0.0000
     Firing Density     20.0000     20.0000     20.0000     20.0000     20.0000
     Temperature        40.0000     30.0000     20.0000     10.0000      5.0000
     Gain               40.0000     30.0000     20.0000     10.0000      5.0000
     Gain               40.0000     30.0000     20.0000     10.0000      5.0000
   Learn Step             99999           0           0           0           0
     Coefficient 1       1.0000      0.0000      0.0000      0.0000      0.0000
     Coefficient 2       0.0010      0.0000      0.0000      0.0000      0.0000
     Coefficient 3       0.0000      0.0000      0.0000      0.0000      0.0000
     Temperature        10.0000      0.0000      0.0000      0.0000      0.0000
   PE: A
       1.000 Err Factor      0.000 Desired
       0.002 Sum             1.000 Transfer          1.000 Output
           6 Weights         0.000 Error            12.000 Current Error
   PE: B
       1.000 Err Factor      0.000 Desired
       0.006 Sum             1.000 Transfer          1.000 Output
           6 Weights         0.000 Error            22.000 Current Error
   PE: C
       1.000 Err Factor      0.000 Desired
       0.049 Sum             1.000 Transfer          1.000 Output
           6 Weights         0.000 Error            17.000 Current Error
   PE: D
       1.000 Err Factor      0.000 Desired
       0.031 Sum             1.000 Transfer          1.000 Output
           6 Weights         0.000 Error             7.000 Current Error
Layer: OUTPUT
        PEs: 1        Wgt Fields: 1                  Sum: Sum
    Spacing: 5         F' offset: 0.00           Transfer: StepFunction
      Shape: Square                              Output: Direct
      Scale: 1.00     Low Limit: -9999.00     Error Func: standard
     Offset: 0.00    High Limit: 9999.00           Learn: --None--
   Init Low: 0.000    Init High: 0.000      L/R Schedule: (Network)
   Winner 1: None                             Winner 2: None
   PE: 15
       1.000 Err Factor      0.000 Desired
       4.000 Sum             1.000 Transfer          1.000 Output
           4 Weights         0.000 Error            -1.000 Current Error
```

```
Title: Boltzman Network - FIG 24 -Output Data
       Display Mode: Network                          Type: Hetero-Associative
      Display Style: default
   Control Strategy: boltzio                  L/R Schedule: default
          0 Learn                27 Recall                  2 Layer
         10 Aux 1                10 Aux 2                  20 Aux 3
L/R Schedule: default
   Recall Step                 1          0          0          0          0
     Input Clamp          1.0000     0.0000     0.0000     0.0000     0.0000
     Firing Density     100.0000     0.0000     0.0000     0.0000     0.0000
     Temperature          0.0000     0.0000     0.0000     0.0000     0.0000
     Gain                 1.0000     0.0000     0.0000     0.0000     0.0000
     Gain                 1.0000     0.0000     0.0000     0.0000     0.0000
   Learn Step              99999          0          0          0          0
     Coefficient 1        1.0000     0.0000     0.0000     0.0000     0.0000
     Coefficient 2        0.0000     0.0000     0.0000     0.0000     0.0000
     Coefficient 3        0.0000     0.0000     0.0000     0.0000     0.0000
     Temperature          0.0000     0.0000     0.0000     0.0000     0.0000
IO Parameters
         Learn Data: File Rand. (FIG24TRN) Binary Load
        Recall Data: File Seq. (FIG24)
        Result File: Desired Output, Output
     UserIO Program: userio
         I/P Ranges:    -1.0000,     1.0000
         O/P Ranges:     0.0000,     1.0000
      I/P Start Col:          1              MinMax Table: default
      O/P Start Col:          5         Number of Entries: 0
MinMax Table <default>:
Layer: 1
        PEs: 1       Wgt Fields: 2                  Sum: Sum
    Spacing: 5        F' offset: 0.00             Transfer: Linear
      Shape: Square                                Output: Direct
      Scale: 1.00     Low Limit: -9999.00      Error Func: standard
     Offset: 0.00    High Limit: 9999.00           Learn: --None--
   Init Low: -0.100   Init High: 0.100       L/R Schedule: (Network)
   Winner 1: None                               Winner 2: None
   PE: Bias
       1.000 Err Factor      0.000 Desired
       0.000 Sum             1.000 Transfer         1.000 Output
           0 Weights         0.000 Error            0.000 Current Error
Layer: INPUT
        PEs: 4       Wgt Fields: 1                  Sum: Sum
    Spacing: 5        F' offset: 0.00             Transfer: StepFunction
      Shape: Square                                Output: Direct
      Scale: 1.00     Low Limit: -9999.00      Error Func: standard
     Offset: 0.00    High Limit: 9999.00           Learn: --None--
   Init Low: -0.100   Init High: 0.100       L/R Schedule: (Network)
   Winner 1: None                               Winner 2: None
   PE: 1
       1.000 Err Factor     22.000 Desired
      22.000 Sum             1.000 Transfer         1.000 Output
           0 Weights        22.000 Error            0.000 Current Error
   PE: 2
       1.000 Err Factor     17.000 Desired
      17.000 Sum             1.000 Transfer         1.000 Output
           0 Weights        17.000 Error            0.000 Current Error
   PE: 3
       1.000 Err Factor      7.000 Desired
       7.000 Sum             1.000 Transfer         1.000 Output
           0 Weights         7.000 Error            0.000 Current Error
   PE: 4
       1.000 Err Factor      7.000 Desired
       7.000 Sum             1.000 Transfer         1.000 Output
           0 Weights         7.000 Error            0.000 Current Error
```

```
Layer: BOLTMANN
        PEs: 4        Wgt Fields: 2                  Sum: Sum
    Spacing: 5         F' offset: 0.00           Transfer: Boltzmann
      Shape: Square                              Output: Direct
      Scale: 1.00     Low Limit: -9999.00     Error Func: standard
     Offset: 0.00    High Limit: 9999.00           Learn: Boltzmann
   Init Low: 0.000    Init High: 0.000      L/R Schedule: boltzman
   Winner 1: None                           Winner 2: None
L/R Schedule: boltzman
   Recall Step                 6          11          16          21          26
     Input Clamp          0.0000      0.0000      0.0000      0.0000      0.0000
     Firing Density      20.0000     20.0000     20.0000     20.0000     20.0000
     Temperature         40.0000     30.0000     20.0000     10.0000      5.0000
     Gain                40.0000     30.0000     20.0000     10.0000      5.0000
     Gain                40.0000     30.0000     20.0000     10.0000      5.0000
   Learn Step              99999           0           0           0           0
     Coefficient 1        1.0000      0.0000      0.0000      0.0000      0.0000
     Coefficient 2        0.0010      0.0000      0.0000      0.0000      0.0000
     Coefficient 3        0.0000      0.0000      0.0000      0.0000      0.0000
     Temperature         10.0000      0.0000      0.0000      0.0000      0.0000
   PE: A
       1.000 Err Factor      0.000 Desired
       0.043 Sum             1.000 Transfer          1.000 Output
           5 Weights         0.000 Error            22.000 Current Error
   PE: B
       1.000 Err Factor      0.000 Desired
       0.032 Sum             1.000 Transfer          1.000 Output
           5 Weights         0.000 Error            17.000 Current Error
   PE: C
       1.000 Err Factor      0.000 Desired
      -0.032 Sum             1.000 Transfer          1.000 Output
           5 Weights         0.000 Error             7.000 Current Error
   PE: D
       1.000 Err Factor      0.000 Desired
       0.002 Sum             1.000 Transfer          1.000 Output
           5 Weights         0.000 Error             7.000 Current Error
Layer: OUTPUT
        PEs: 1        Wgt Fields: 1                  Sum: Sum
    Spacing: 5         F' offset: 0.00           Transfer: StepFunction
      Shape: Square                              Output: Direct
      Scale: 1.00     Low Limit: -9999.00     Error Func: standard
     Offset: 0.00    High Limit: 9999.00           Learn: --None--
   Init Low: 0.000    Init High: 0.000      L/R Schedule: (Network)
   Winner 1: None                           Winner 2: None
   PE: 15
       1.000 Err Factor      0.000 Desired
       4.000 Sum             1.000 Transfer          1.000 Output
           4 Weights         0.000 Error            -1.000 Current Error
```

FIG. 32 INFORMATION

```
! FIG. 32 Test Data

1000 45 34 21 5   4   34 23 9 0 37 23 -1000 -45 -34 -21 -5    -4   -34 -23 -9 0 -37 -23
1000 45 34 21 5   9   34 23 4 0 37 23 -1000 -45 -34 -21 -5    -9   -34 -23 -4 0 -37 -23
1000 5  34 21 45 4   34 23 9 0 37 23 -1000 -5  -34 -21 -45   -4   -34 -23 -9 0 -37 -23
1000 5  10 21 45 32 34 7  9 0 5  23 -1000 -5  -10 -21 -45   -32 -34 -7  -9 0 -5  -23
1000 5  10 21 45 32 22 7  9 0 5  23 -1000 -5  -10 -21 -45   -32 -22 -7  -9 0 -5  -23
1000 5  10 21 45 3  22 7  9 0 34 23 -1000 -5  -10 -21 -45   -3  -22 -7  -9 0 -34 -23
1000 5  10 24 45 13 19 7  9 0 5  23 -1000 -5  -10 -24 -45 -13  -19 -7  -9 0 -5  -23
1000 5  10 24 45 13 19 7  9 0 15 23 -1000 -5  -10 -24 -45 -13 - 19 -7  -9 0 -15 -23
```

FIG. 33 INFORMATION

```
! FIG. 33 Training Data

-1 -1 -1 0 0 0 -1 0 -1 0 -1 -1
& 1 0 0 0 1 1 0 0 0 1 0 0
-1 -1 -1 0 0 0 -1 0 0 0 -1 -1
& 1 0 0 0 1 0 0 0 1 1 0 0
-1 0 -1 0 -1 0 -1 0 -1 0 -1 -1
& 1 1 0 0 0 1 0 0 0 1 0 0
-1 0 0 0 -1 -1 -1 0 0 0 0 -1
& 1 1 1 0 0 0 1 0 0 1 1 0
-1 0 0 0 -1 -1 -1 0 0 0 0 -1
& 1 1 1 1 0 0 0 1 0 1 1 1
-1 0 -1 0 -1 0 -1 0 -1 0 -1 -1
& 1 1 1 1 0 1 1 1 0 1 0 0
-1 0 0 -1 -1 -1 0 0 0 0 0 -1
& 1 1 1 0 0 0 1 0 0 1 1 0
-1 0 0 -1 -1 -1 1 1 1 -1 1 1
& 1 1 1 0 0 1 1 0 0 1 0 0
```

FIG. 32 INFORMATION

```
! FIG. 32 Test Data- Changing position of start and goal cells

1000 45 34 21 5 4 34    23 9     0  37 23 -1000 -45 -34 -21 -5 -4 -34 -23 -9     0  -3
1000 0  34 21 5 4 34    23 9     12 37 23 -1000  0  -34 -21 -5 -4 -34 -23 -9    -12 -3
1000 45  0 21 5 4 34    23 9     12 37 23 -1000 -45   0 -21 -5 -4 -34 -23 -9    -12 -3
1000 45 34 0  5 4 34    23 9     12 37 23 -1000 -45 -34   0 -5 -4 -34 -23 -9    -12 -3
1000 45 34 21 0 4 34    23 9     12 37 23 -1000 -45 -34 -21  0 -4 -34 -23 -9    -12 -3
1000 45 34 21 5 0 34    23 9     12 37 23 -1000 -45 -34 -21 -5  0 -34 -23 -9    -12 -3
0    45 34 21 5 4 1000 23 9     12 37 23  0    -45 -34 -21 -5 -4 -1000 -23 -9 -12 -3
0    45 34 21 5 4 34 1000 9     12 37 23  0    -45 -34 -21 -5 -4 -34 -1000 -9 -12 -3
0    45 34 21 5 4 34    23 1000 12 37 23  0   -45 -34 -21 -5 -4 -34 -23 -1000 -12 -3
0    45 34 21 5 4 34    23 9  1000 37 23  0    -45 -34 -21 -5 -4 -34 -23 -9 -1000 -3
15   45 34 21 5 4 1000 23 9     12 37  0  -15  -45 -34 -21 -5 -4 -1000 -23 -9 -12 -3
15 1000 34 21 0 4 34    23 9     12 37 23  -15 -1000 -34 -21 0 -4 -34  -23 -9 -12 -37

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

FIG. 35 INFORMATION

```
! FIG. 35 Training Data - Changing percent cells filled with obstacles

-1 -1 -1 0 0 0 -1 0 -1 0 -1 -1
& 1 0 0 0 1 0 0 0 1 1 0 0
-1 -1 -1 0 0 0 -1 0 0 0 -1 -1
& 1 0 0 0 1 0 0 0 1 1 0 0
-1 0 0 0 -1 -1 -1 0 0 0 0 -1
& 1 1 0 0 0 1 0 0 0 1 0 0
-1 0 -1 0 -1 0 -1 0 -1 0 -1 -1
& 1 1 0 0 0 1 0 0 0 1 0 0
-1 0 0 -1 -1 -1 0 0 0 0 0 -1
& 1 1 0 0 0 1 0 0 0 1 0 0
-1 0 0 -1 -1 -1 1 1 1 -1 1 1
& 1 1 0 0 0 1 0 0 0 1 0 0

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

FIG. 37 INFORMATION

```
! FIG. 37 Training Data

-1 -1 0 0 0 -1 0 -1 0 -1 -1
& 1 0 0 0 1 1 0 0 0 1 0 0
-1 0 -1 0 0 0 -1 0 -1 0 -1 -1
& 1 1 0 0 0 0 0 0 0 0 0 0
-1 -1 0 0 0 0 -1 0 -1 0 -1 -1
& 1 0 1 0 1 1 1 0 0 0 0 0
-1 -1 -1 0 0 0 -1 0 -1 0 -1 -1
& 1 0 0 1 1 1 1 1 0 0 0 0
-1 -1 -1 0 0 0 -1 0 -1 0 -1 -1
& 1 0 0 0 1 0 0 0 0 0 0 0
-1 -1 -1 0 0 0 -1 0 -1 0 -1 -1
& 1 0 0 0 1 1 0 0 0 0 0 0
0 -1 -1 0 0 0 -1 0 -1 0 -1 -1
& 1 0 0 0 1 1 1 0 0 0 0 0
0 -1 0 0 0 0 -1 -1 -1 0 0 -1
& 1 0 0 0 1 1 1 1 0 0 0 0
0 -1 -1 0 0 0 -1 0 -1 0 -1 -1
& 1 0 0 0 1 0 0 0 1 0 0 0
0 -1 -1 0 0 0 -1 0 -1 -1 -1 0
& 0 0 0 0 0 0 1 1 0 0 0 1
-1 -1 -1 0 0 0 -1 0 -1 -1 -1 0
& 0 1 0 0 1 1 0 0 0 0 0 0
-1 -1 -1 0 0 0 -1 0 -1 0 -1 -1
& 0 1 0 0 1 1 0 0 0 0 0 0
```

NOTES

[1] Shahir, Micha. "Algorithmic Motion Planning in Robotics." Computer, 22, No.3, March 1989, p.11, col. 3, para. 2.

[2] Lozano-Perez, Tomas. Robot Motion: Planning and Control. pp. 499-535. Ed. Michael Brady, John M. Hollerbach, Timothy L. Johnson, Tomas Lozano-Perez and Matthew T. Mason. Massachusetts: The MIT Press, 1982.

[3] Fritsch, F.N. "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles." Communications of the ACM. 22, No. 10, October 1979, pp. 562,col. 1,para. 1.

[4] Fritsch, F.N. "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles." Communications of the ACM. 22, No. 10, October 1979, pp. 562,col. 2,para. 2.

[5] Lozano-Perez, Tomas. Robot Motion: Planning and Control. pp. 473-497. Ed. Michael Brady, John M. Hollerbach, Timothy L. Johnson, Tomas Lozano-Perez and Matthew T. Mason. Massachusetts: The MIT Press, 1982. p.486, para. 5.

[6] Lozano-Perez, Tomas. Robot Motion: Planning and Control. pp. 473-497. Ed. Michael Brady, John M. Hollerbach, Timothy L. Johnson, Tomas Lozano-Perez and Matthew T. Mason. Massachusetts: The MIT Press, 1982. p.489, para. 3.

[7] Lozano-Perez, Tomas. Robot Motion: Planning and Control. pp. 499-535. Ed. Michael Brady, John M. Hollerbach, Timothy L. Johnson, Tomas Lozano-Perez and Matthew T. Mason. Massachusetts: The MIT Press, 1982. p. 516, para. 2.

[8] Lozano-Perez, Tomas. Robot Motion: Planning and Control. pp. 499-535 and pp. 473-497. Ed. Michael Brady, John M. Hollerbach, Timothy L. Johnson, Tomas Lozano-Perez and Matthew T. Mason. Massachusetts: The MIT Press, 1982.

[9] Faverjon, B., P. Tournassoud. "The Mixed Approach for Motion Planning: Learning Global Strategies from a Local Planner." Proceedings of the Tenth International Joint Conference on Artificial Intelligence. pp. 1131-1137. Ed. John McDermott. SanFrancisco: Interprint, 1987.

[10] Faverjon, B., P. Tournassoud. "The Mixed Approach for Motion Planning: Learning Global Strategies from a Local Planner." Proceedings of the Tenth International Joint Conference on Artificial Intelligence. pp. 1131-1137. Ed. John McDermott. SanFrancisco: Interprint, 1987. p. 1133, col. 1, para. 3.

[11] Wasserman, P.. Neural Computing Theory and Practice. pp. 18, Figure 1-5, New York: Van Nostrand Reinhold, 1989.

[12] Wasserman, P. Neural Computing Theory and Practice. pp. 31, Figure 2-5 and Table 2-1, New York: Van Nostrand Reinhold, 1989.

[13] Wasserman, P. Neural Computing Theory and Practice. pp. 35, Figure 2-8, New York: Van Nostrand Reinhold, 1989.

[14] Wasserman, P. Neural Computing Theory and Practice. pp. 46, Figure 3-3, New York: Van Nostrand Reinhold, 1989.

[15] Wasserman, P. Neural Computing Theory and Practice. pp. 79, Figure 5-2, New York: Van Nostrand Reinhold, 1989.

[16] Wasserman, P. Neural Computing Theory and Practice. pp. 83, Figure 5-3, New York: Van Nostrand Reinhold, 1989.

[17] Wasserman, P. Neural Computing Theory and Practice. pp. 95, Figure 6-1, New York: Van Nostrand Reinhold, 1989.

[18] NeuralWare, Reference Guide, Neuralworks Professional II/Plus and Neuralworks Explorer. pp. RF-184 and RF-185, Pennsylvania: Neuralware, 1991.

[19] NeuralWare, Reference Guide, Neuralworks Professional II/Plus and Neuralworks Explorer. pp. RF-183, para. 4 Pennsylvania: Neuralware, 1991.

[20] Hassoun, Mohamad H., Sanghvi, Ashvin J. "Fast Computation of Optimal Paths in Two- and Higher Dimension Maps." In 1990 IEEE Transactions on Neural Networks. Vol. 3, No. 3, pp.355-363., 1990.

BIBLIOGRAPHY

Akiyama, Yutaka., Tamashira, Akira., Mashahiro, Kajiura., Anzai, Yuichiro., Aiso, Hideo. "The Gaussian Machine: A Stochastic Neural Network for Solving Assignment Problems." Journal of Neural Network Computing. Technology, Design, and Applications. Vol. 2, Number 3, pp. 43-551., 1991.

Borenstein, Johann., Koren, Yorem., "Real-Time Obstacle Avoidance for Fast Mobile Robots." IEEE Transactions on Systems, Man, and Cybernetics. Vol. 19, Number 5., pp. 1179-1187., September/October 1989.

Faverjon, B., P. Tournassoud. "The Mixed Approach for Motion Planning: Learning Global Strategies from a Local Planner." Proceedings of the Tenth International Joint Conference on Artificial Intelligence. pp. 1131-1137. Ed. John McDermott. SanFrancisco: Interprint, 1987.

Fritsch, F.N. "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles." Communications of the ACM. 22, No. 10, October 1979, pp. 560-570.

Georgeff, Michael P., Amy L. Lansky. "Reactive Reasoning and Planning." Proceedings AAAI-87 Sixth National Conference on Artificial Intelligence. Vol. 2, 1987, pp. 677-682. California: Morgan Kaufman Publishers, Inc., 1987.

Gevarter, William B., Intelligent Machines: An Introductory Perspective of Artificial Intelligence and Robotics. pp. 15-29 and pp. 67-81. New Jersey: Prentice-Hall, Inc., 1985.

Goldberg, David E., Genetic Algorithms in Search and Optimization. Massachusetts: Addison-Wesley Publishing Company Inc., 1989.

Grattinger, T.J., Krigh, B.H. "Evaluation and Time-Scaling of Trajectories for Wheeled Mobile Robots." Transactions of the ASME. Vol. 111 , pp. 222-231., June 1989.

Hassoun, Mohamad H., Ashvin J. Sanghvi. "Fast Computation of Optimal Paths in Two- and Higher Dimension Maps." In 1990 IEEE Transactions on Neural Networks. Vol. 3, No. 3, pp. 355-363., 1990.

Khouri, J., K.A. Stelson. "An Efficient Algorithm for Shortest Path in Three Dimensions with Polyhedral Obstacles." Transactions of the ASME. 3, No. 3, September 1989, pp. 433-436.

Kohn, W., Van Valkenburg, J.A., Dunn, C.K. "Automatic Procedures Generator for orbital rendezuous maneuver." SPIE Space Station Automation, Vol. 580, pp. 40-52, 1985.

Korf, R.E. "Real-Time Heuristic Search." Artificial Intelligence. An International Journal. Vo. 42, Numbers 2-3, pp.189-211, March 1990.

Lozano-Perez, Tomas, wesley, Micheal A. "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obtacles." Communications of the ACM. Vol. 22, Number 10, pp.560-570, October 1979.

Lozano-Perez, Tomas. Robot Motion: Planning and Control. pp. 499-535 and pp. 473-497. Ed. Michael Brady, John M. Hollerbach, Timothy L. Johnson, Tomas Lozano-Perez and Matthew T. Mason. Massachusetts: The MIT Press, 1982.

NeuralWare Inc. Neural Computing, Neuralworks Professional II/Plus and NeuralWorks Explorer. Pennsylvania : NeuralWare Inc. Technical Publications Group, 1991.

NeuralWare Inc. Reference Guide NeuralWorks Professional II/ Plus and NeuralWorks Explorer. Pennsylvania : NeuralWare Inc. Technical Publications Group, 1991.

NeuralWare Inc. Using NeuralWorks, An Extended Tutorial for NeuralWorks Professional II/Plus and NeuralWorks Explorer. Pennsylvania : NeuralWare Inc. Technical Publications Group, 1991.

Noboria, Hiroshi., Naniwa, Tomohide., Arimoto, Suguru "A Quadtree-Based Path-Planning Algorithm for a Mobile Robot." Journal of Robotic Systems. Vol. 7, Number 4, pp. 555-574., August 1990.

Rich, Elaine. Artificial Intelligence. pp. 247-277. New York: McGraw-Hill Company, 1983.

Ritter, Helge J., Martinetz, Thomas M., Schulten, Klaus J. "Topology-Conserving Maps for Learning Visuo-Motor-Coordination." Neural Networks, Vol. 2, Number 3, pp. 159-168., 1989.

Schwartz, Jacob T., Sharir, Micha "On the Piano Mover's Problem: I. The case of a Two-Dimensional Rigid Polygonal Body Moving Admist Polygonal Barriers." Planning, Geometry, and Complexity of Robot Motion. pp. 1-50, 1987.

Shahir, Micha. "Algorithmic Motion Planning in Robotics." Computer; 22, No. 3, March 1989, pp. 9-20.

Tychonievich, Lou., David Zaret, John Mantenga, Robert Evans, Eric Muehle, Scott Martin. "A Maneuvering-Board Approach to Path Planning with Moving Obstacles." Proceedings of the Eleventh International Joint Conference on Artificial Intelligence. Vol. 2, pp. 1017-1021. Ed. N.S. Sridharan. California: Morgan Kaufman Publishers, Inc., 1989.

Wasserman, Philip D., Neural Computing Theory and Practice. New York: Van Nostrand Rheinhold., 1989.