# Neurosolver Solves Blocks World Problems

**Andrzej Bieszczad**
*andrzej@carleton.ca*

**Bernard Pagurek**
*bernie@carleton.ca*

Systems and Computer Engineering, Carleton University
Ottawa, Colonel By Drive, CANADA K1S 5B6

*ABSTRACT*

*We report an ongoing work on a biologically inspired device, the Neurosolver, introduced in our earlier papers. To explore and improve the Neurosolver's capabilities, we attempt to apply it to a task of rearranging three different blocks in a blocks world similar to the worlds known from the classic AI literature. The Neurosolver records the observed trajectories in the state space of the blocks world and uses the learned traces to perform searches and construct plans to control the movements of the blocks. This work is a part of a broader effort to devise neurally-based agents that would cooperate en masse in a process of solving complex problems. We consider it a next step toward a Neuromorphic General Problem Solver.*

## 1. Introduction

The goal of the research that led to the original introduction of the Neurosolver, as reported in (Bieszczad, [2]) and (Bieszczad, [3]), was to design a neuromorphic device that would be able to tackle problems in the framework of the state space paradigm (Newell, [13]). The research was inspired by Burnod's monograph on the workings of the human brain (Burnod, [13]). Although the current architecture is well rooted in the previous work, the Neurosolver has evolved considerably since its introduction. The modifications were critical for improving the practical aspects of the behavior of the Neurosolver.

The class of the systems that employ state spaces to present and solve problems has its roots in the early stages of the AI research derived from the studies of human information processing, e.g. on General Problem Solver (Newell, [13]). This pioneer work led to very interesting problem solving (e.g. SOAR, (Laird, [9])) and planning systems (e.g. STRIPS, (Nillson, [12])). NeuroSOAR (Cho, [6]) is one of the few attempts to explore similar ideas from the neural network perspective, although its main focus is on how SOAR's production system and decision procedure can be implemented in neural networks. There were other attempts to use neural networks in the context of robot path planning (see (Connolly, [7]) for a short review). Higher level tasks were addressed by rule-based connectionists models, e.g. (Touretzky, [17]). Temporal difference models (Sutton, [16]), with a very successful application in playing backgammon (Tesauro, [17]), are interesting in the context of state spaces, because they can be derived from Markov models (Barnard, [1]). An interesting research on merging logic and connectionists systems is reported in (Sun, [15]). The work on processing temporal sequences (a general taxonomy can be found in (Mozer, [11])) is concerned mostly with recognition of temporal patterns, but such systems can also be used to generate sequences (e.g., predict the future).

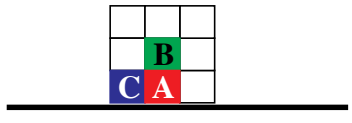



**FIGURE 1. A block world with a 3x3 workspace and three blocks: red, green and blue.**

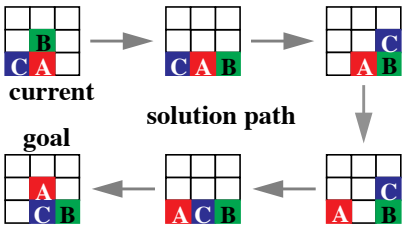
**FIGURE 2. The solution to the problem in a**

To the best of our knowledge, the Neurosolver is unique with its close links to the classic AI paradigm and a direct use of a state space for performing searches and generating problem solutions.

To explore the capabilities of the Neurosolver, we created a version of the blocks world that was used by many researchers in the past, e.g. (Nillson, [12]). The world, as illustrated in Figure 1, consists of three blocks that can be moved in a workspace of nine positions arranged in three columns and three rows. A block can be in any position, providing it has either the table or another block beneath. The front end to the blocks world includes a user interface that allows the operator to move the blocks, compose sequences of moves, save and restore sets of teaching samples and control the learning and the behavior of the Neurosolver. The Neurosolver interacts with the blocks world through a transparent interface that provides an easy way to make modifications on both ends. Additional displays are available to observe the activity patterns, input vectors and to control state changes by pointing and clicking at nodes.

## 2. Problem state space

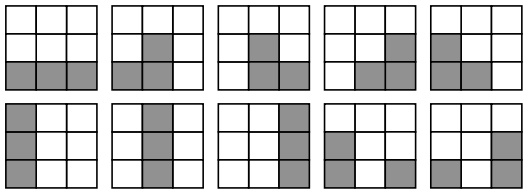
There are many types of problems in the blocks world. To keep things simple, we consider the problem of rearranging the blocks from an initial configuration to a desired configuration. Any block configuration in the blocks world constitutes a state of the system. Changes in time to the configuration are described in terms of state transitions, i.e., sequences of two states corresponding to two consecutive configurations, and trajectories, i.e., sequences of more states that can be decomposed into a series of individual state transitions. A state transition spans two adjacent time instants, while a trajectory is a history of the system behavior over a longer time interval. A problem to solve is posed by a pair of states: the goal and the current state. The solution to the problem is a trajectory starting with the current state and ending with the goal (Figure 2). Only valid state transitions can be included in the solution.




**global sense is a sequence of configurations between the current configuration (state) and the goal.**

**FIGURE 3. There are 10 possible**

Taking into account the constraints on the blocks and disregarding color, there are 10 arrangements of the blocks as illustrated in Figure 3. Each three blocks can be permuted in 6 ways, so if the color is taken into account again, there are 10x6 possible configurations. Therefore, the cardinality of the problem state space is 60. In this relatively simple example, it is possible to determine the size of the problem state space by a straightforward analysis.



**arrangements of blocks if the color is disregarded. Three block can be arranged in 6 ways, there are 10x6 possible configurations taking into account the three colors.**

**FIGURE 4. Architecture of the Neurosolver**

With more complex problems, however, it may be difficult or not possible at all to predict which states should be accounted for. In such a case, Kohonen self-organizing maps (Kohonen, [15]) can be used to eliminate states that are not valid, because the states that are not used, and invalid states are not used, are not present in the topology map. The Kohonen algorithm is a clustering technique that in its extremal version results in a one-to-one mapping of input into output. One can start with a large number of available outputs nodes and end up only with some of them being actually used for storing states, because some inputs may never be activated. The unused outputs can be pruned to limit the size of the network. Some of the unused nodes may be left for the future to accommodate any inputs that have not been accounted for so far, but may be activated later on. A supervised algorithm, for example the backpropagation or Kohonen's LVQ, could also be used, but an unsupervised algorithms are superior in the sense that they do not need any a priori knowledge and can adapt autonomously; i.e., without a teacher.

## 3. The Neurosolver

### 3.1. General Architecture

The Neurosolver is a network of interconnected nodes. Each node is associated with a state in a problem space. In this work, that association was pre-wired, but any of the existing associative algorithms could be used in general case; e.g., LVQ (Kohonen, [8]) or backpropagation (Rumelhart, Hinton, Williams, [14]). In fact, there are two networks that, while providing different functionality, cooperate in the resolution of problems. A problem is presented to the Neurosolver by two signals: the goal, i.e. the limbic input[1] to the node associated with the desired state, and the sensory, i.e., the thalamic[2] input to the node associated with the current state. A sequence of firing nodes represents a trajectory in the state space. A solution to a problem is, then, a succession of firing nodes starting with the current node and ending with the goal node.Let us introduce the architecture of the node first, since it will make it easier to explain how the both networks of the Neurosolver are built and how they work.

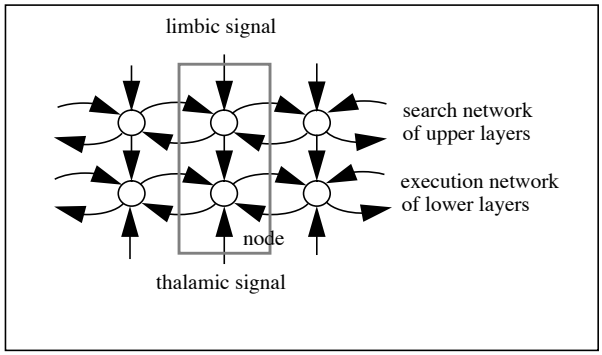
---

1. The limbic system of the human brain is believed to be the source of wishes and desires; that is the origin of the name for the limbic input

2. The thalamus is a relay system between the environment perceived by the sensors and the brain.

### 3.2. The Node

The node used in the Neurosolver is based on a biological cortical column (the references to the relevant neurobiological literature can be found in (Bieszczad, [2])). It consists of two divisions: the upper and the lower, as illustrated in Figure 4. The upper division is a device integrating internal signals from other upper divisions and from the control center providing the limbic input; i.e., a goal or, using more psychological terms, a drive or a wish. The activity of the upper division is transmitted to the lower division where it is subsequently integrated with signals from other lower divisions and the thalamic input.The connectivity between the nodes is also biologically inspired, but to increase the efficiency by limiting the number of objects to process, the internodal connections are constructed dynamically during the learning process, so only the connections with a non-null strength are present.




The input function of the upper division is a sum of the products of the action potentials of all afferent connections and their strengths and the limbic input. The input function to the lower division similarly integrates the incoming action potentials with the thalamic input.

Both, the upper and the lower divisions use threshold functions as their output function. If the activity of a division is lower then the threshold, no action potential is generated in the efferent connections. The only exception is the connection between the upper and lower divisions of the same node. The strength of that connection is 1, but a threshold function is applied by the lower division to this input with a threshold that is lower than the one used by the output function. This ensures that the activity of the upper division alone cannot activate the lower division to the level exceeding the threshold of the output function of the lower division; i.e., it cannot fire the node.

The output of the lower division of the node is that node's output.

### 3.3. Search network

The upper divisions of the nodes and the connections between them constitute the first of the two mentioned earlier networks. The function of this network is to spread the activity along the connections, starting at the node associated with the goal state that receives the limbic input, the original source of activity. This is a search network, because the activity is spread in a hope that at some node it will be integrated within the lower division receiving the thalamic input to the level exceeding the output threshold. In that moment, a path from the goal to the current state has been found. This is a solution path (there may be many solution paths).

A tree created by the pattern of spreading activity is called a call tree, following the convention used by Burnod.

### 3.4. Execution network

The second network is composed of the lower divisions of the nodes and their connections. Its purpose is to generate output signals along the solution path by firing successive nodes. Only one node is allowed to fire at any time; all other nodes are suppressed. A node that fires sends actions potentials through the outgoing connections of its lower division. These signals may cause another node to fire, if the expectation, or attention, of that node, that is the activity in the upper division, is high enough. In a way, thanks to the winner-takes-all policy, the process of selecting the successor in the resolution path is a form of selecting the node that is activated the most.

After firing, the lower division of the node is inhibited for some time, so immediate oscillations are avoided. The upper division is still working as usual, because the inhibited node may be a relay station for important branches of the call tree. Inhibiting the node completely might cut off some important clues.

### 3.5. Learning state transitions

The learning by the Neurosolver is probabilistic. Examples are presented by providing thalamic inputs to corresponding nodes. During the learning, each such signal forces firing of the node. The number of firings, F, is recorded in the node and used later to detect cycles.

For each state transition, two connections are strengthened: one, in the direction of the transition, between the lower divisions of the two nodes, and another, in the opposite direction, between the upper divisions. The connection between the upper and lower divisions of the same node is fixed. The number of transmissions of an action potential, T, is recorded for each connection. For each division, two statistical data are collected: the total number of cases when

the division positively influenced other nodes, $S_{out}$, and the total number of cases when the division positively responded to the stimuli from others, $S_{in}$. A positive influence means that a firing node sent an action potential to a node that fired next.

Using the statistics, the Neurosolver determines the strength of each connection. The strength is computed as a product of two probabilities: the probability that a firing source node will generate an action potential in the connection and the probability that the target node will fire upon receiving the action potential from the connection:

$$C = \frac{T}{S(source)_{out}} \cdot \frac{T}{S(target)_{in}} \qquad \text{(EQ 1)}$$

### 3.6. Solving problems

The limbic input to the node associated with the goal state activates the upper division of the node. That activity spreads along the learned trajectories in the direction that is opposite to the state transitions from the learning samples due to the way the upper-to-upper connections are constructed in the learning process. The limbic input is the source of the activity for as long as the goal node fires, in which case the node is shut down. The goal node will fire if a sequence of nodes fires between the nodes associated with the current state and the goal. This chain reaction can be triggered at the node associated with the current state when its lower division gets sufficient activity from the upper division and the thalamic input to exceed the firing threshold of the node; i.e., the output threshold of the lower division. The next node in the solution sequence fires providing that the activity in the upper division and the input from the lower division of the node that fired just before also sum up to a level exceeding the firing threshold.

This is not guaranteed, so a hard imposed cut-off is in place on a number of time ticks of a process that takes too long. If that is the case, then no solution can be found and the system responds with a failure. Before the cut-off limit is used, however, there are a number of steps taken that increase the chance of finding a solution. Firstly, the limbic input of the goal node is raised gradually up to the level just below the firing threshold; i.e., a stronger desire to resolve the problem is introduced. In that way, the overall activity of the call tree is increased, so some new branches may be explored, because the activity in some leaves may now exceed the search threshold; i.e., the output threshold of upper divisions. If that process still does not fire a node, then the search threshold is gradually lowered. The effect of this action is similar the former: the call tree gets larger. As the final resort, the firing threshold is gradually lowered to increase a chance of finding a firing a node. Only if all of that fails, the Neurosolver quits without finding a solution. If a node fires, the drive is reset back to its initial value. The following pseudo-code illustrates that process:

```
procedure solve;
    while networkActive and not goalAchieved and not cycleDetected do
    begin
        while networkActive and not goalAchieved and not cycleDetected do
            search;
        if driveLevel < UpperBoundOnDriveLevel then
            driveLevel := enforce(driveLevel);          /* e.g., enf(x)=k*x or enf(x)=x+x^2  */
            setLimbicInput(goalNode, driveLevel);
            networkActive := true;
        else
            if searchThreshold > LowerBoundOnUpperThreshold then
                searchThreshold := searchThreshold - ActivityDelta;
                networkActive := true;
            else
                if firingThreshold > LowerBoundOnLowerThreshold then
                    firingThreshold := firingThreshold - ActivityDelta;
                    networkActive := true;
    end;
```

The thresholds, as well as the initial strengths of the limbic and thalamic inputs and the progress step are the parameters that can be used to control the speed and granularity of the resolution.

Activity spreading networks have been used by others, e.g. (Maes, [10]), but we believe that the way we apply the ideas in the context of problem solving is novel.

# 4. Neurosolver in the blocks world

## 4.1. Interface

### 4.1.1 Input representation

The interface to the Neurosolver allows for straightforward interaction between the blocks world application and the Neurosolver. The state of the blocks world is translated into a binary vector that is presented to the Neurosolver. Each location of the block world is represented by two input nodes:

- 00 - empty
- 01 - red (A)
- 10 - green (B)
- 11 - blue (C).

There are (3x3)x2=18 input nodes. Any configuration of the blocks can be expressed using that number of nodes.

The input nodes are connected to state nodes in such a way that one state node is activated for each block configuration. The connections are hard-wired, but could be determined by the Kohonen algorithm.

### 4.1.2 Output representation

In the opposite direction, whenever a node fires, the state associated with that node is translated into a binary representation of the block configuration, analogous to the input vector. The blocks in the workspace are re-arranged accordingly.

### 4.1.3 Performance

During the experiments it was discovered that the Neurosolver exhibited the same capabilities in lower dimensions as it did in larger state spaces, with a visible difference in processing time, as one may expect. The difference in the speed of learning was not as dramatic as in the search and the execution times. A number of problem spaces with sizes varying from 5 to 200 have been tried. The experiments were started with a small state space involving a network of 5 nodes. That allowed for a careful analysis of the behavior of the Neurosolver and, consecutively, led to many refinements to the architecture and the algorithms. In some cases, the Neurosolver exhibited an ability to synthesize plans as it constructed a solution path by utilizing parts of several learned trajectories.

The following two experiments are detailed in here to show that the capacity of the Neurosolver is considerable. The cardinality of the state space was 60, since no transient states were allowed for these experiments.

Experiment 1

| | | |
|---|---|---|
| Purpose: | to test the capacity of the Neurosolver | |
| Learning sample: | all valid state transitions; each transition presented once | |
| Problem: | all tasks of rearranging the blocks (3600 in total) | |
| Results: | Successes: 3349(~93%) | Failures: 251(~7%) |

Experiment 2

| | | |
|---|---|---|
| Purpose: | to test the capacity of the Neurosolver | |
| Learning sample: | all valid state transitions presented randomly in 10000 iterations | |
| Problem: | all tasks of rearranging the blocks (3600 in total) | |
| Results: | Successes: 3254(~90.3%) | Failures: 346(~9.7%) |

Although the Neurosolver could learn many of the patterns, it has not been able to record all valid trajectories in the blocks world. With many patterns recorded, the problems that had short solution paths could be found in most cases. However, some of the more complex problems were not solved. These problems could finally be solved if fewer patterns were presented during learning or if the trajectory was learned again. Another issue is the quality of the solutions. Some of the more complex problems were solved, but the solutions included cycles. Most of the attempts involving cycles, however, ultimately failed.

## 4.2. Conclusions and Future Work

As mentioned before, the intended use of the Neurosolver is the brain of agents that attempt to resolve blocks world problems in a co-operative manner (Bieszczad, [4]). The hope was to record all valid trajectories in one neurosolver and use it as a uniform controller in every agent. As it turns out, some heuristic approach must be taken to the problems that cannot be solved. Taking into account very high success rate, probably the easiest technique is to move the block to another location hoping that the new configuration can trigger a solution. Another approach could be to use a number of agents controlled by neurosolvers that are specialized for certain tasks.

There are still a number of ideas that may improve the behavior of the Neurosolver. One of them is an introduction of secondary connections that would connect nodes that are further away in the same path (corresponding to higher order Markov models). That would provide a momentum for the movement along the learned path. Unfortunately, such an addition will be a further, substantial strain on the processing resulting in a certain decrease in the speed.

The co-operation of the agents is a challenge. Some channels must be put in place for both, inter-agent communication and links to the external world. After a solution is found, the overall plan must be decomposed into tasks that may be handled by individual agents. There is a question of having agents with a complete knowledge or of sharing that knowledge between the agents and one or more control centers.

The Neurosolver recalculates the activity of each node in each time tick, therefore an implementation on an analog computer or in hardware would be more natural and by far more efficient than a software simulation. That could also address the scaling to more complex applications.

## Acknowledgments

This work is supported in part by Telecommunication Research Institute of Ontario.

## References

[1] Barnard, E. (1993), *Temporal-Difference Methods and Markov Models*, IEEE Transactions on Systems, Man and Cybernetics, vol. 23, no. 2, pp. 357-365.

[2] Bieszczad, A. (1994a), *Neurosolver: a neural network based on a cortical column,* Proceeding of the World Congress on Neural Networks, Vol. II, pp. 756-761, San Diego, CA, INNS Press, Lawrence Erlbaum Associates, New York.

[3] Bieszczad, A. (1994b), *Neurosolver: A Step Toward a Neuromorphic General Problem Solver*, Proceedings of IEEE World Congress on Computational Intelligence, vol. 3, p. 1313-1318, Orlando, FL.

[4] Bieszczad, A. and Pagurek, B. (1995), *Neurosolver-Based Agents for Solving Blocks World Problems*, Proceedings of The First International Conference on Computational Intelligence and Neurosciences, Wrightsville Beach, NC, Elsevier Publishing Co., New York.

[5] Burnod, Y. (1988), *An Adaptive Neural Network: The Cerebral Cortex*, Paris, France: Masson.

[6] Cho, B., Rosenbloom, P. S. and Dolan, C. P. (1991), *Neuro-Soar: A neural network architecture for goal-oriented behavior*, Proceedings of Thirteenth Annual Conference of the Cognitive Science Society, Chicago IL: Lawrence Erlbaum Associates.

[7] Connolly, C. L. (1993), *A Robotics Perspective on Motor Programs and Path Planning*, Behavioral and Brain Sciences, vol. 15, no. 4, pp. 728-729.

[8] Kohonen, T. (1988), *Self-organization and associative memory*, 2nd edition, Berlin, Germany: Springer-Verlag.

[9] Laird, J. E., Newell, A. and Rosenbloom, P. S. (1987), *SOAR: An architecture for general intelligence*, Artificial Intelligence, vol. 33, pp. 1-64.

[10]Maes, P. (1991), *A Bottom-Up Mechanism for Behavior Selection in an Artificial Creature*, in Meyer, J. A. and Wilson, S. W. (Eds.) From Animals to Animats, Cambridge MA: MIT Press,

[11]Mozer, M. C. (1993), *Neural Net Architectures for Temporal Sequence Processing*, in Weigend, A. and Gershenfeld, N. (Eds.) *Predicting the Future and Understanding the Past*, Redwood City CA: Addison-Wesley Publishing.

[12]Nillson, N. J. (1980), *Principles of Artificial Intelligence*, Palo Alto, CA: Tioga Publishing Company.

[13]Newell, A. and Simon, H. A. (1963), *GPS: A program that simulates human thought*, in Feigenbaum, E. A. and Feldman, J. (Eds.), *Computer and thought*, New York NJ: McGrawHill.

[14]Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986), *Learning Internal Representations by Error Propagation* in *Parallel Distributed Processing*, vol. 1, Rumelhart, D. E., McClelland J. L., Eds., Cambridge, MA: MIT Press.

[15]Sun, R. (1994), *A neural network model of causality*, IEEE Transactions on Neural Networks, Vol. 5, No. 4.

[16]Sutton, R. S. (1988), *Learning to predict by the methods of temporal differences*, Machine Learning, vol. 3, pp. 9-44.

[17]Tesauro, G. (1992), *Practical Issues in Temporal Difference Learning*, Machine Learning, vol. 8, pp. 257-277.

[18]Touretzky, D. and Hinton, G. (1988), *A Distributed Connectionists Production System*, Cognitive Science, vol. 12, no. 3, pp. 423-466.